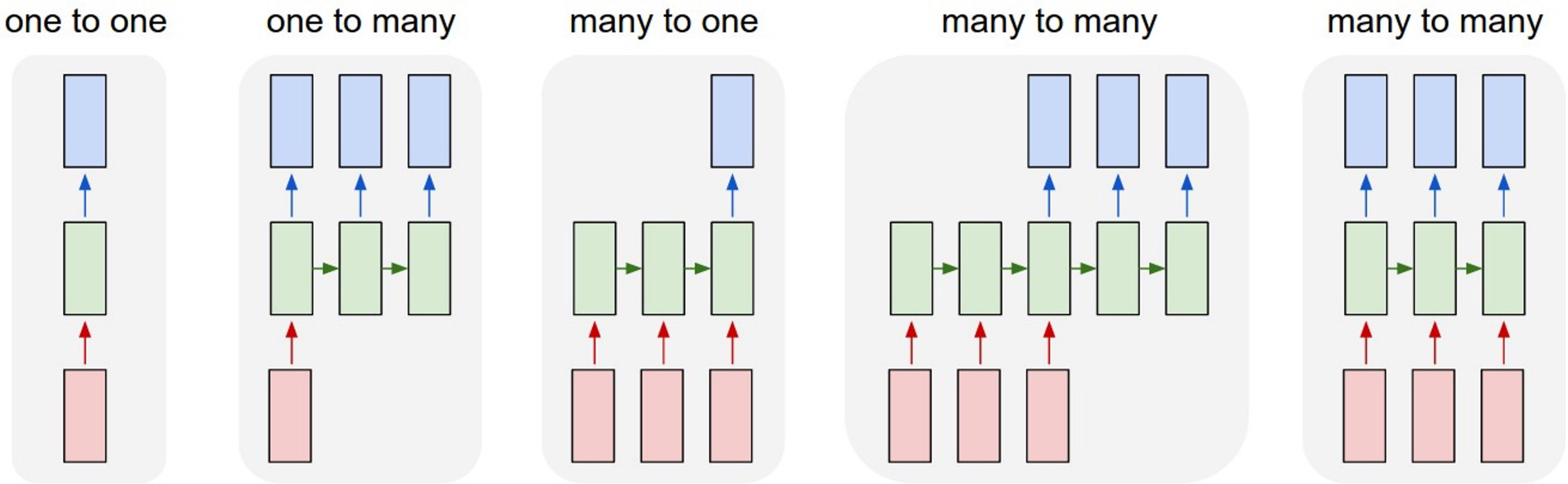
Lecture 17: Attention

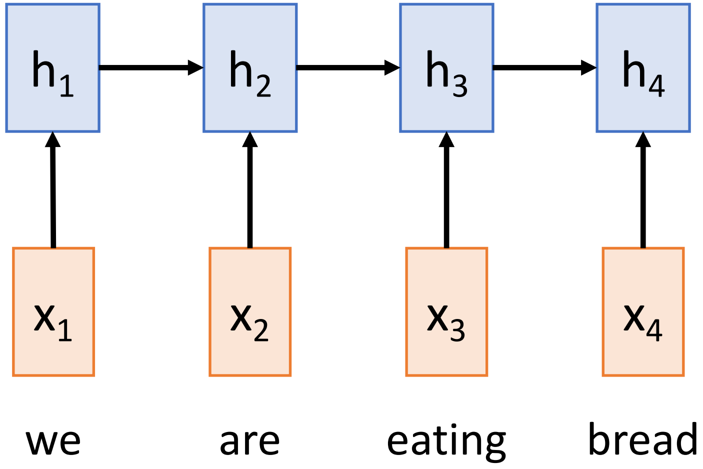
Last Time: Recurrent Neural Networks



Sequence-to-Sequence with RNNs

Input: Sequence $x_1, \dots x_T$
Output: Sequence $y_1, \dots, y_{T'}$

Encoder: $h_t = f_W(x_t, h_{t-1})$



Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

Sequence-to-Sequence with RNNs

Input: Sequence $x_1, \ldots x_T$

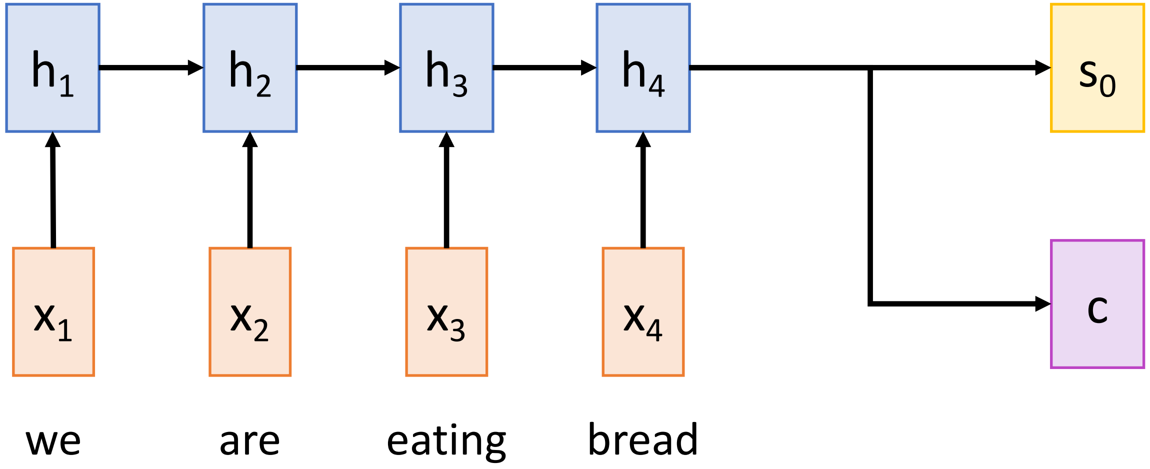
Output: Sequence $y_1, \ldots, y_{T'}$

Encoder: $h_t = f_W(x_t, h_{t-1})$

From final hidden state predict:

Initial decoder state s_0

Context vector c (often $c=h_T$)



Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

Sequence-to-Sequence with RNNs

Input: Sequence $x_1, \dots x_T$

Output: Sequence $y_1, \dots, y_{T'}$

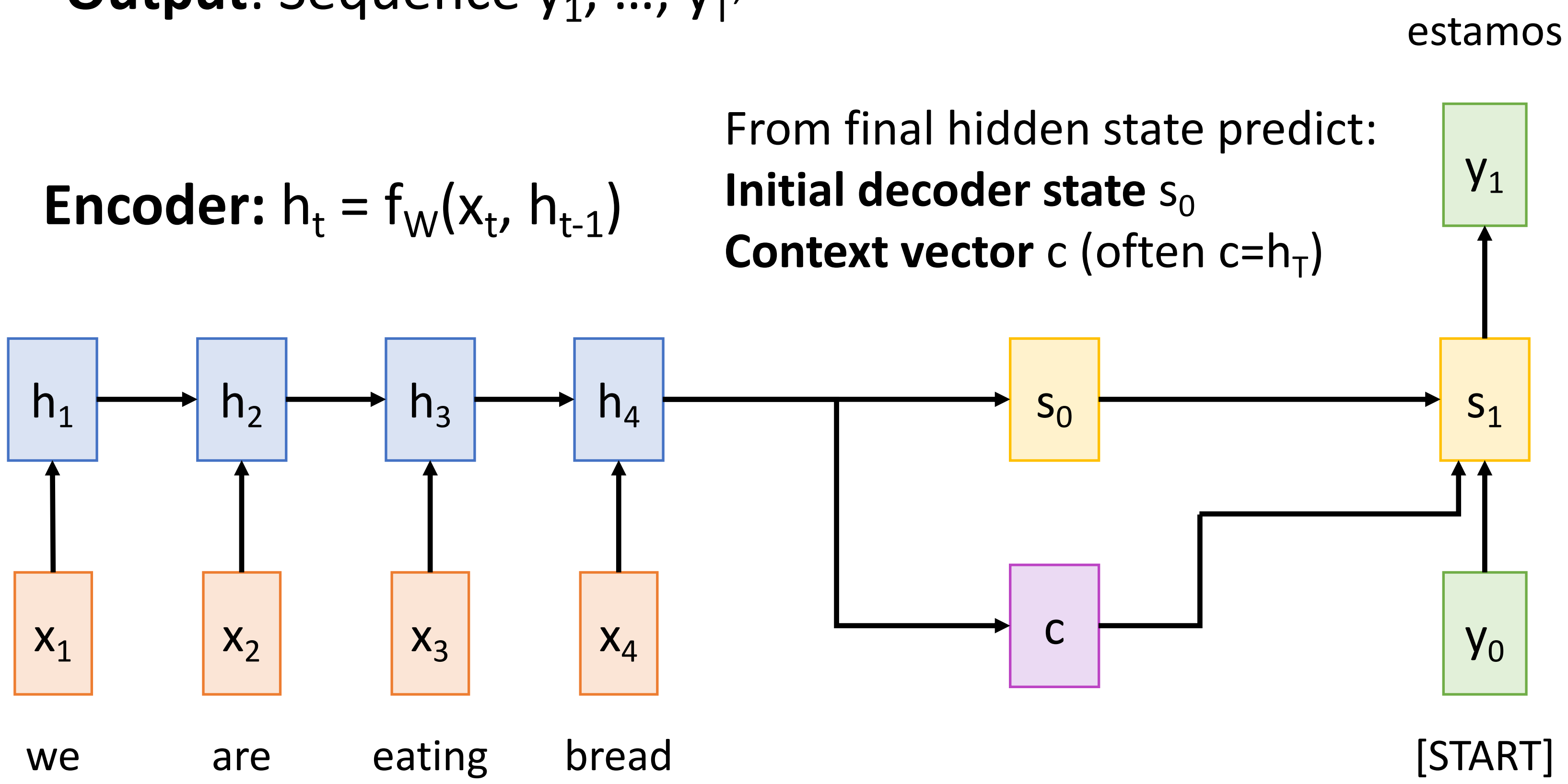
Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

Encoder: $h_t = f_W(x_t, h_{t-1})$

From final hidden state predict:

Initial decoder state s_0

Context vector c (often $c=h_T$)



Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

Sequence-to-Sequence with RNNs

Input: Sequence $x_1, \dots x_T$

Output: Sequence $y_1, \dots, y_{T'}$

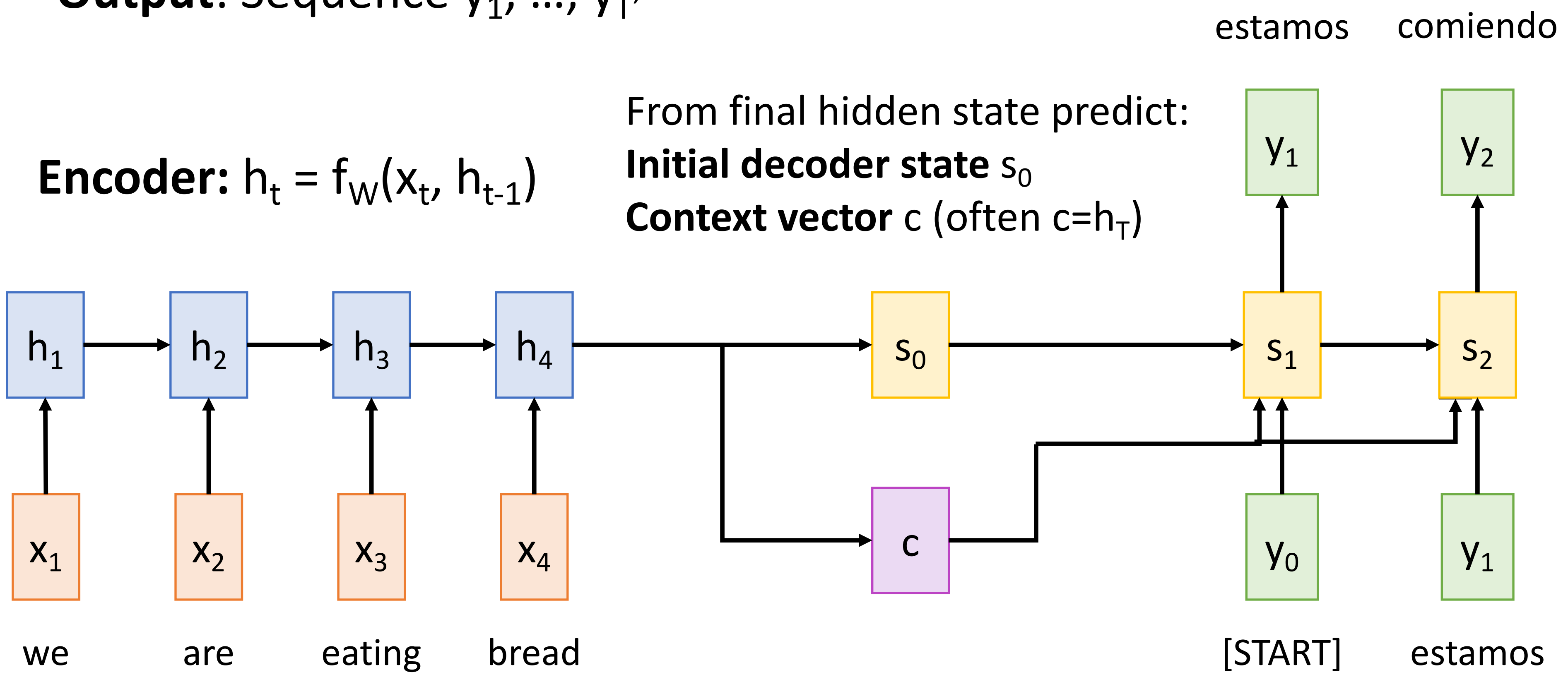
Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

Encoder: $h_t = f_W(x_t, h_{t-1})$

From final hidden state predict:

Initial decoder state s_0

Context vector c (often $c=h_T$)



Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

Sequence-to-Sequence with RNNs

Input: Sequence $x_1, \ldots x_T$

Output: Sequence $y_1, \ldots, y_{T'}$

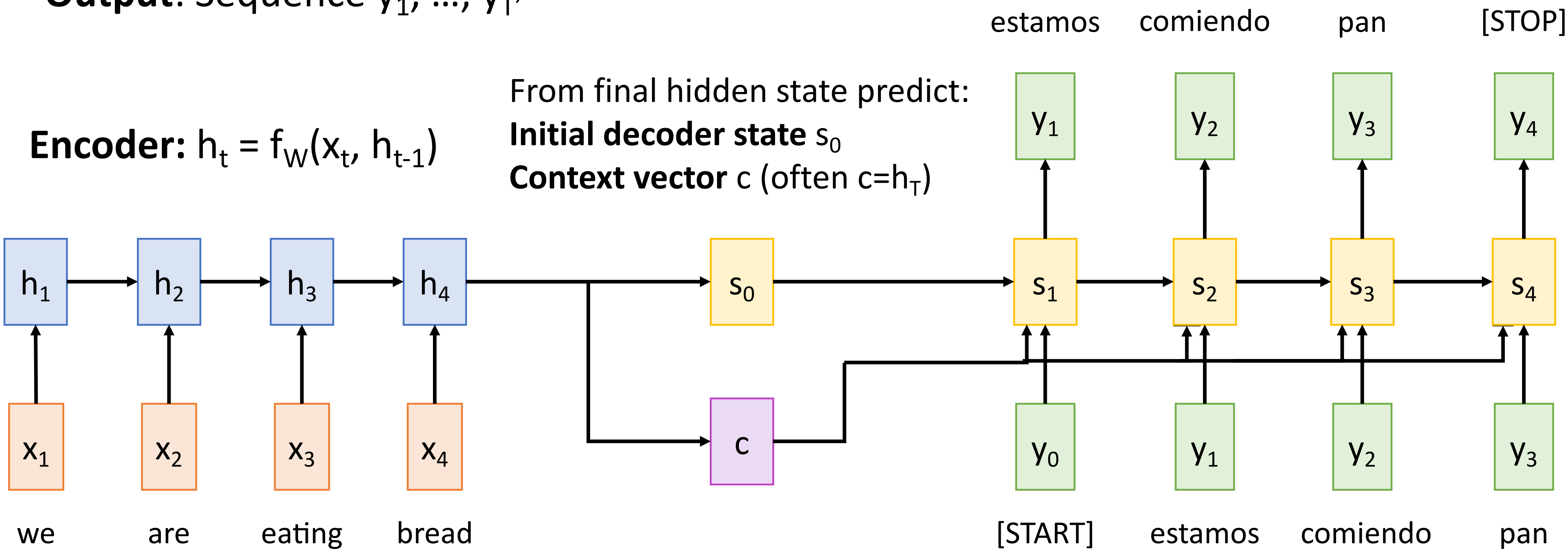
Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

Encoder: $h_t = f_W(x_t, h_{t-1})$

From final hidden state predict:

Initial decoder state s_0

Context vector c (often $c=h_T$)



Sutskever et al, “Sequence to sequence learning with neural networks”, NeurIPS 2014

Sequence-to-Sequence with RNNs

Input: Sequence $x_1, \ldots x_T$

Output: Sequence $y_1, \ldots, y_{T'}$

输入序列的信息被压缩到一个固定大小的向量中，限制了模型的表达能力

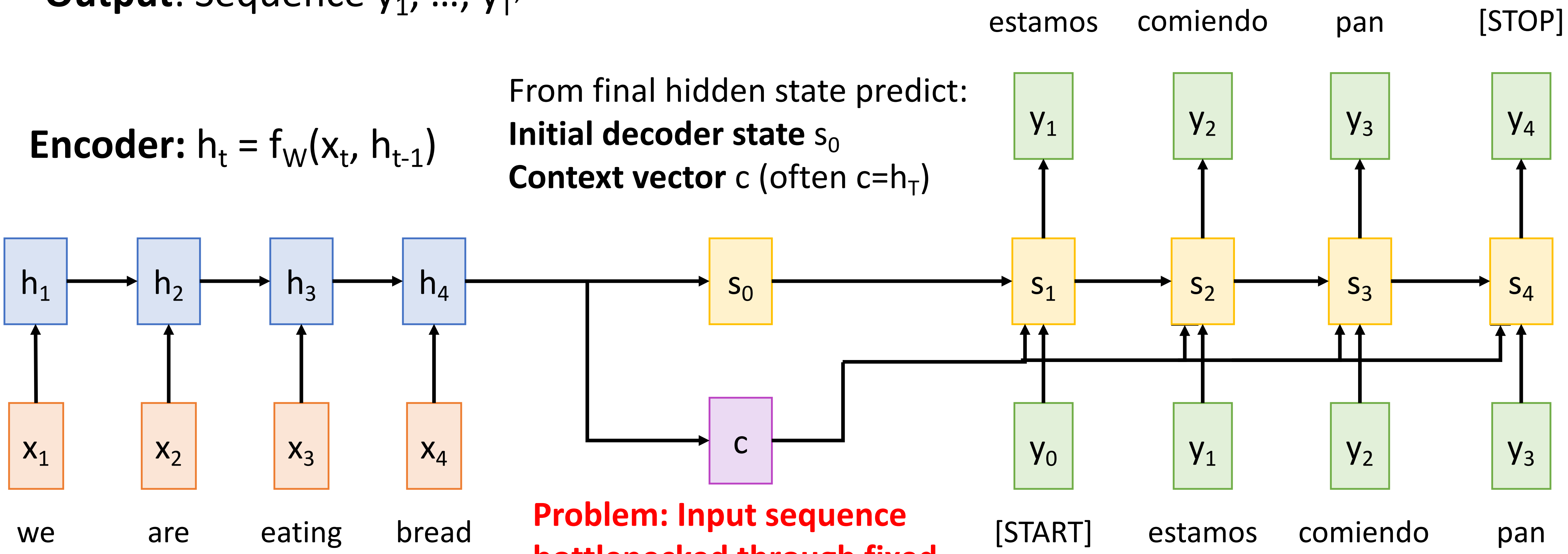
Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

Encoder: $h_t = f_W(x_t, h_{t-1})$

From final hidden state predict:

Initial decoder state s_0

Context vector c (often $c=h_T$)



Problem: Input sequence bottlenecked through fixed-sized vector. What if T=1000?

Sutskever et al, “Sequence to sequence learning with neural networks”, NeurIPS 2014

Sequence-to-Sequence with RNNs

Input: Sequence $x_1, \ldots x_T$

Output: Sequence $y_1, \ldots, y_{T'}$

Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

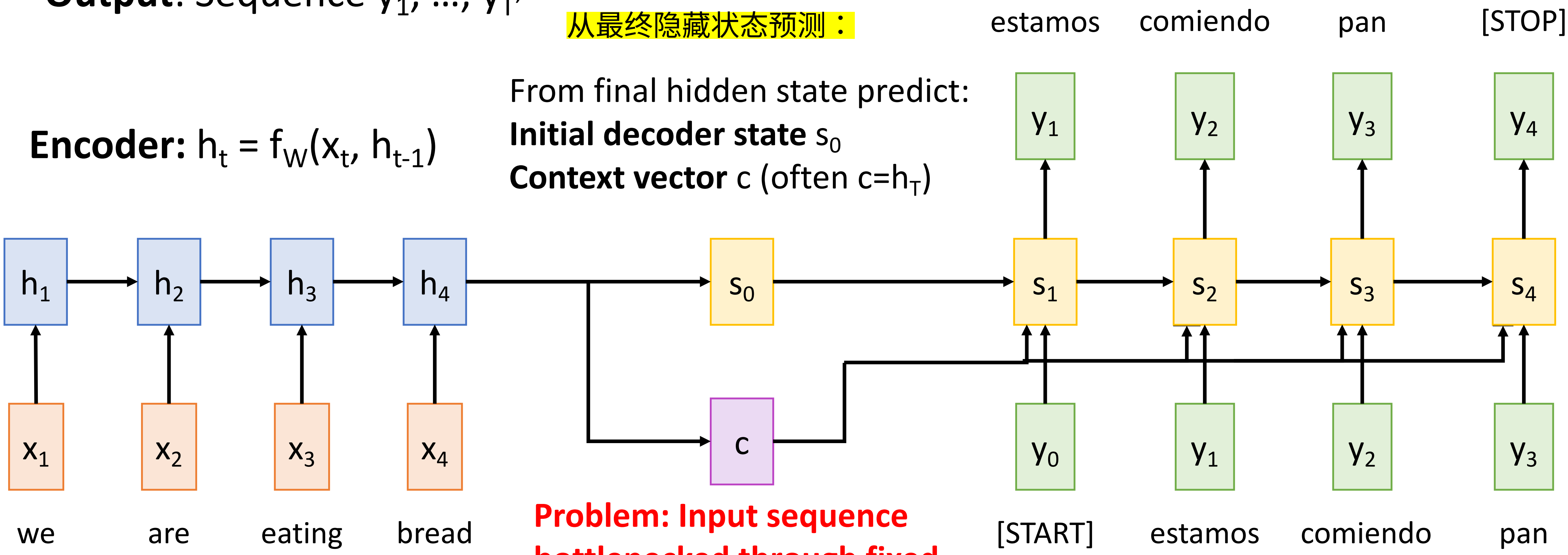
从最终隐藏状态预测：

From final hidden state predict:

Initial decoder state s_0

Context vector c (often $c=h_T$)

Encoder: $h_t = f_W(x_t, h_{t-1})$



Problem: Input sequence bottlenecked through fixed-sized vector. What if T=1000?

Idea: use new context vector at each step of decoder!

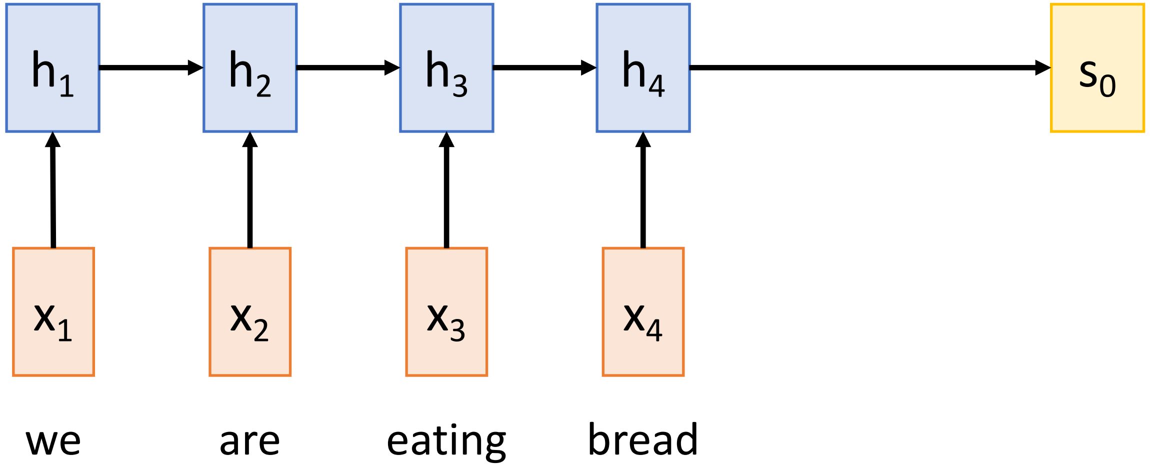
Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

Sequence-to-Sequence with RNNs **and Attention**

Input: Sequence $x_1, \ldots x_T$
Output: Sequence $y_1, \ldots, y_{T'}$

Encoder: $h_t = f_W(x_t, h_{t-1})$

From final hidden state:
Initial decoder state s_0



Q：当要翻译的句子较长时，一个Context 可能存不下那么多信息，就会造成精度的下降。除此之外，如果按照上述方式实现，只用到了编码器的最后一个隐藏层状态，信息利用率低下。

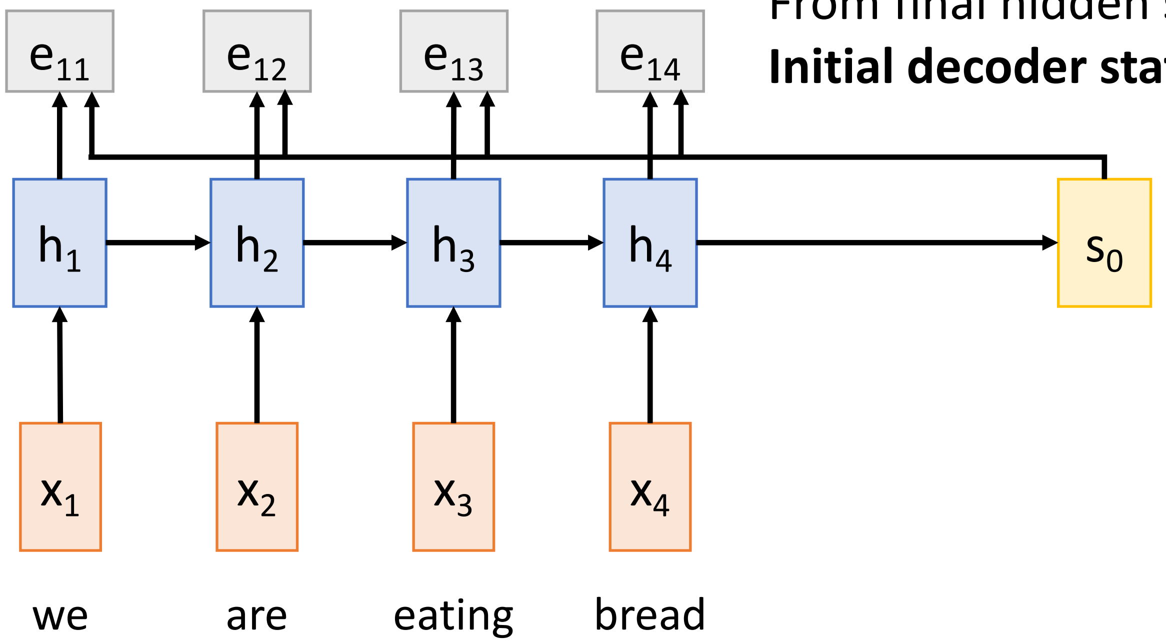
A：利用Encoder所有隐藏层状态解决Context长度限制问题。

Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence-to-Sequence with RNNs **and Attention**

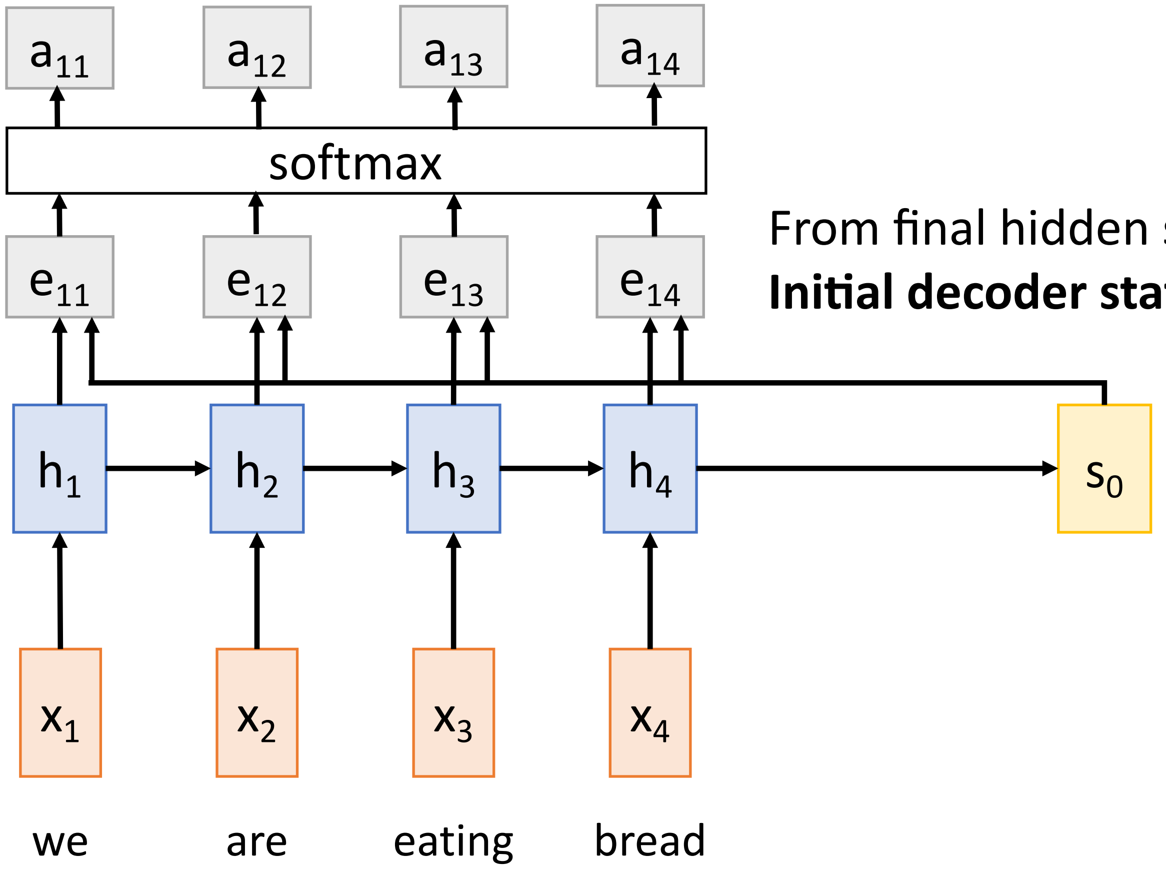
Compute (scalar) **alignment scores**
$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

From final hidden state:
Initial decoder state s_0



Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence-to-Sequence with RNNs **and Attention**



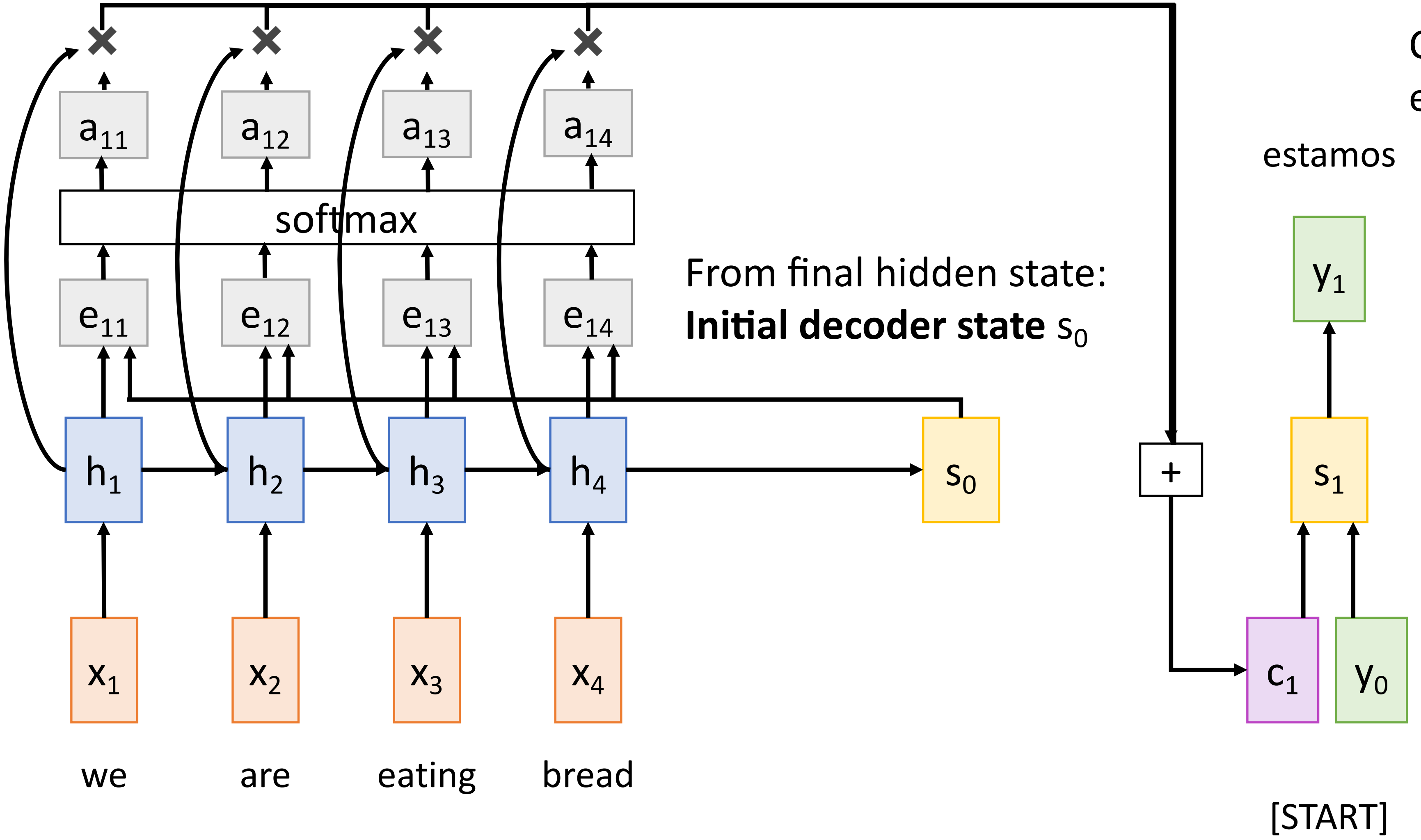
From final hidden state:
Initial decoder state s_0

Compute (scalar) **alignment scores**
$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

Normalize alignment scores to get **attention weights**
$0 < a_{t,i} < 1$ $\sum_i a_{t,i} = 1$

Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence-to-Sequence with RNNs **and Attention**



Compute (scalar) **alignment scores**
$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

Normalize alignment scores to get **attention weights**
$0 < a_{t,i} < 1$ $\sum_i a_{t,i} = 1$

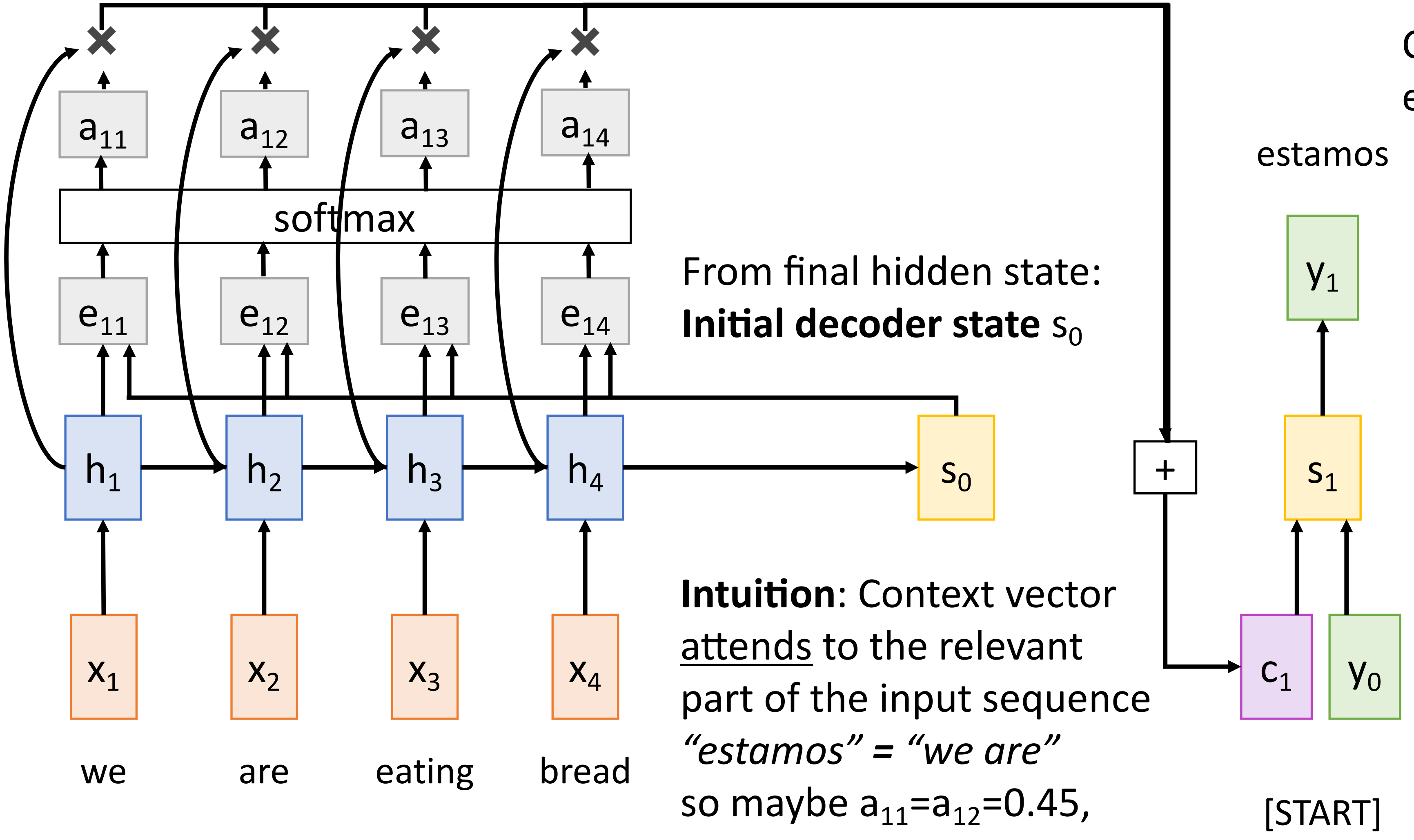
Compute context vector as linear combination of hidden states
$c_t = \sum_i a_{t,i} h_i$

Use context vector in decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c_t)$

This is all differentiable! Do not supervise attention weights – backprop through everything

Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence-to-Sequence with RNNs **and Attention**



Compute (scalar) **alignment scores**
$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

Normalize alignment scores to get **attention weights**
$0 < a_{t,i} < 1 \quad \sum_i a_{t,i} = 1$

Compute context vector as linear combination of hidden states
$c_t = \sum_i a_{t,i} h_i$

Use context vector in decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c_t)$

根据损失函数计算出的梯度来更新网络参数。包括注意力权重，以及网络中的其他所有参数。即前向计算后向调整。

This is all differentiable! Do not supervise attention weights – backprop through everything

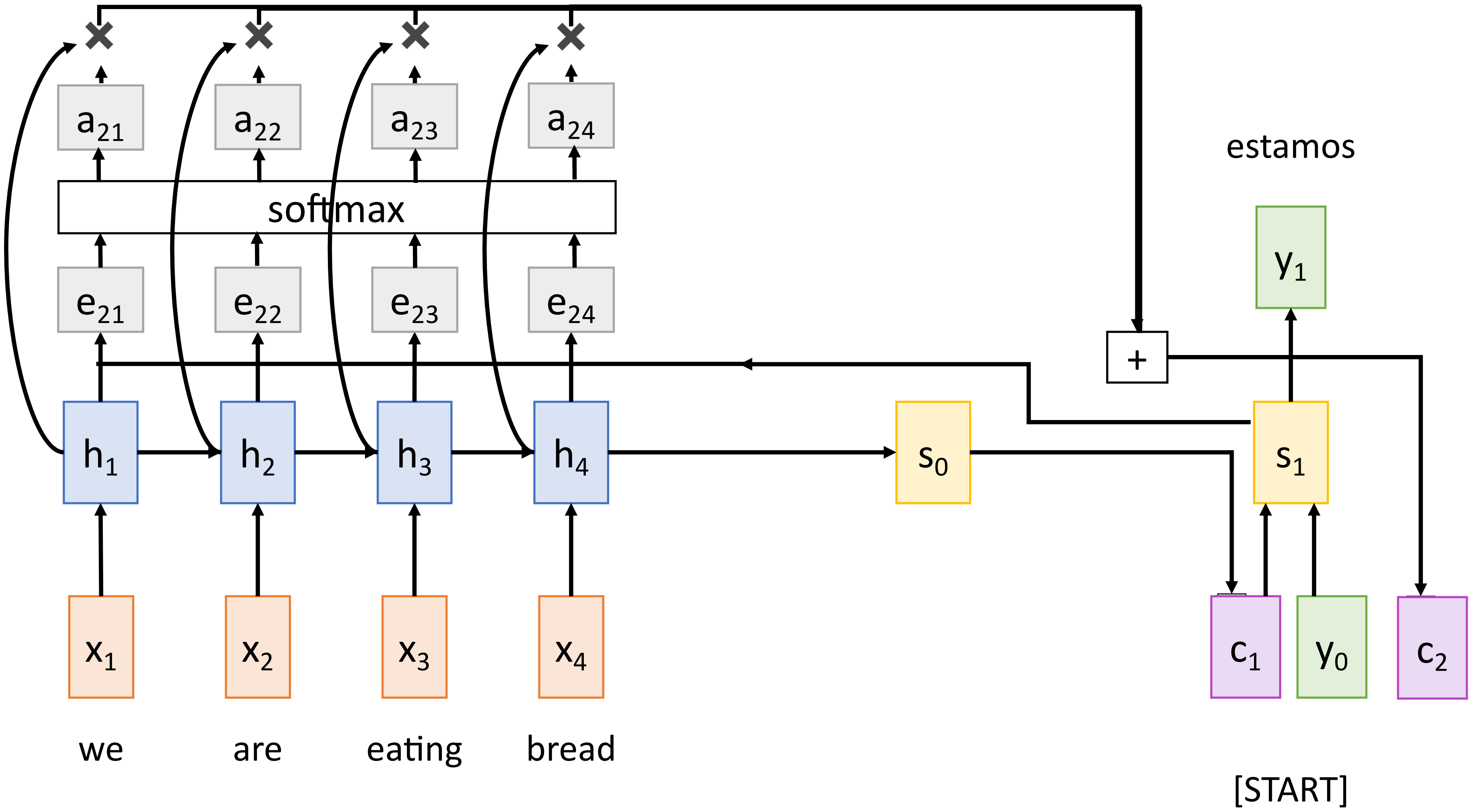
From final hidden state:
Initial decoder state s_0

Intuition: Context vector attends to the relevant part of the input sequence
"estamos" = "we are"
so maybe $a_{11}=a_{12}=0.45$, $a_{13}=a_{14}=0.05$

Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

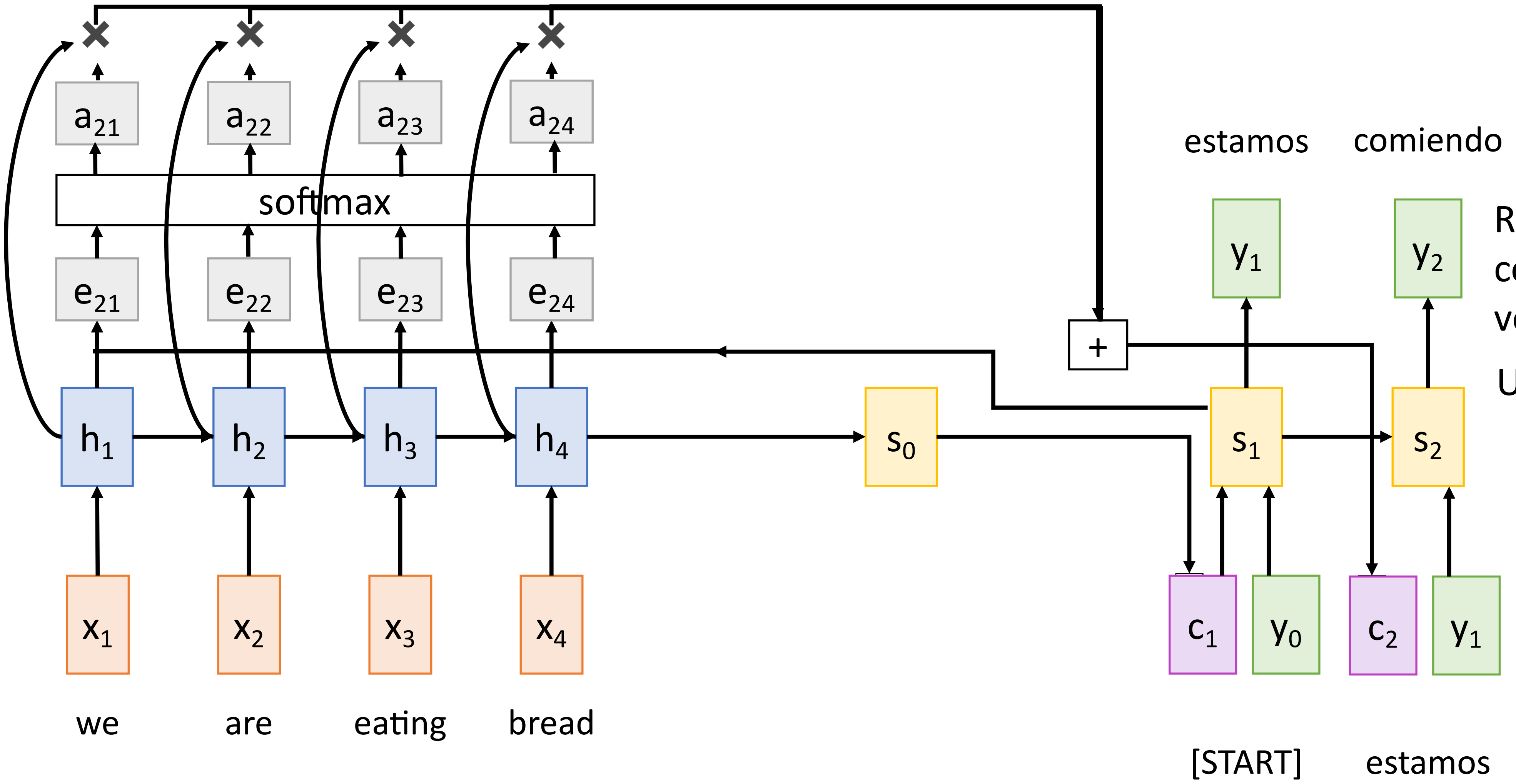
Sequence-to-Sequence with RNNs

Repeat: Use s_1 to compute new context vector c_2



Bahdanau et al, “Neural machine translation by jointly learning to align and translate”, ICLR 2015

Sequence-to-Sequence with RNNs **and Attention**

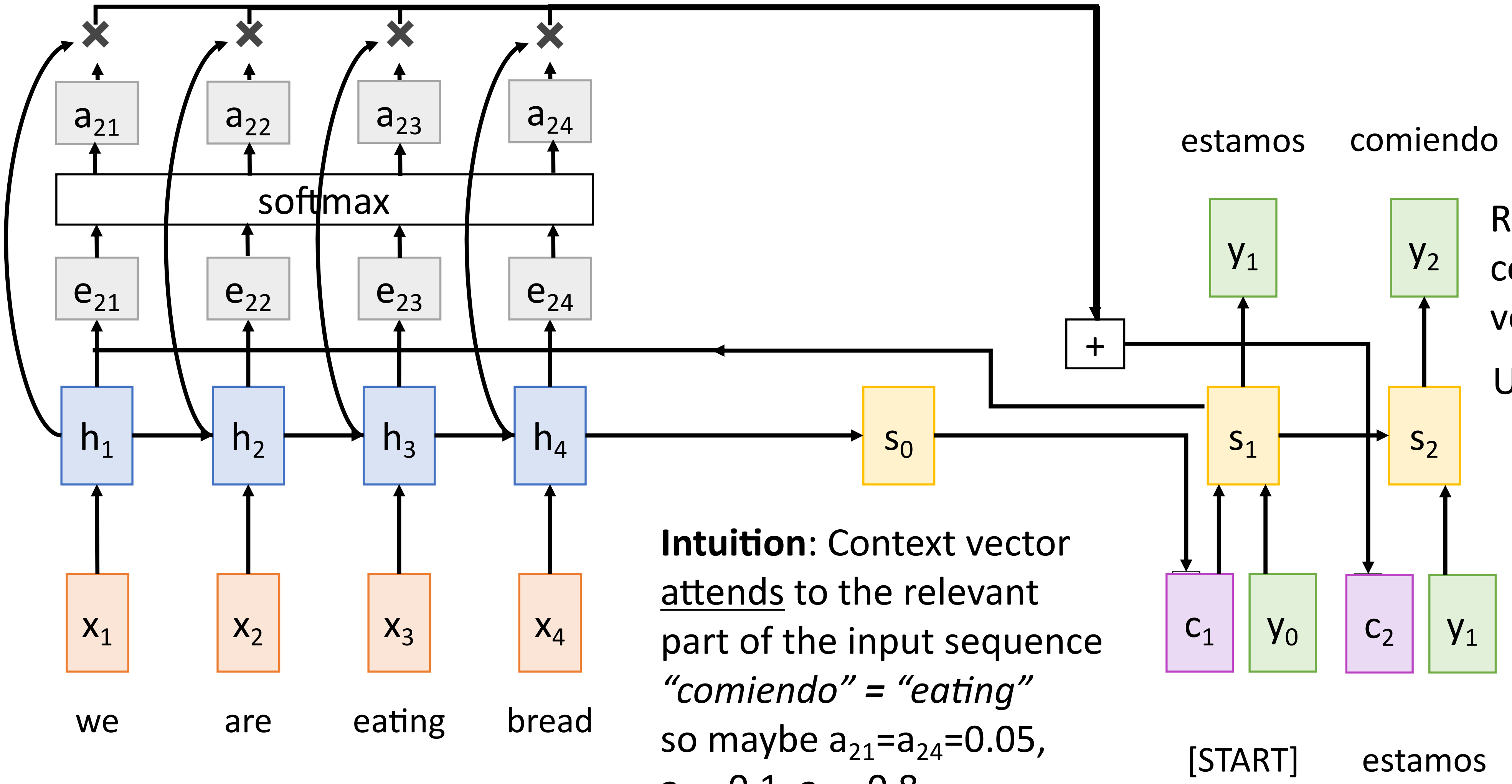


Repeat: Use s_1 to compute new context vector c_2

Use c_2 to compute s_2, y_2

Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence-to-Sequence with RNNs **and Attention**



Repeat: Use s_1 to compute new context vector c_2

Use c_2 to compute s_2, y_2

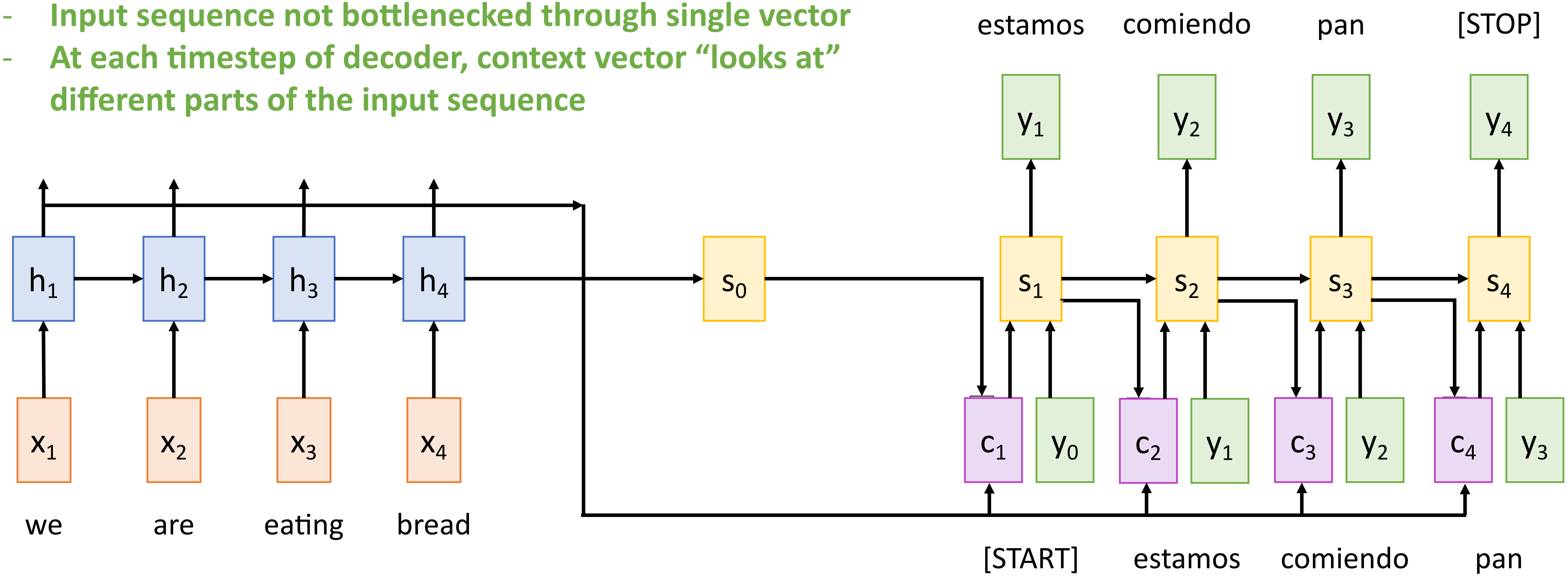
Intuition: Context vector <u>attends</u> to the relevant part of the input sequence
"comiendo" = "eating"
so maybe $a_{21}=a_{24}=0.05$, $a_{22}=0.1$, $a_{23}=0.8$

Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence-to-Sequence with RNNs **and Attention**

Use a different context vector in each timestep of decoder

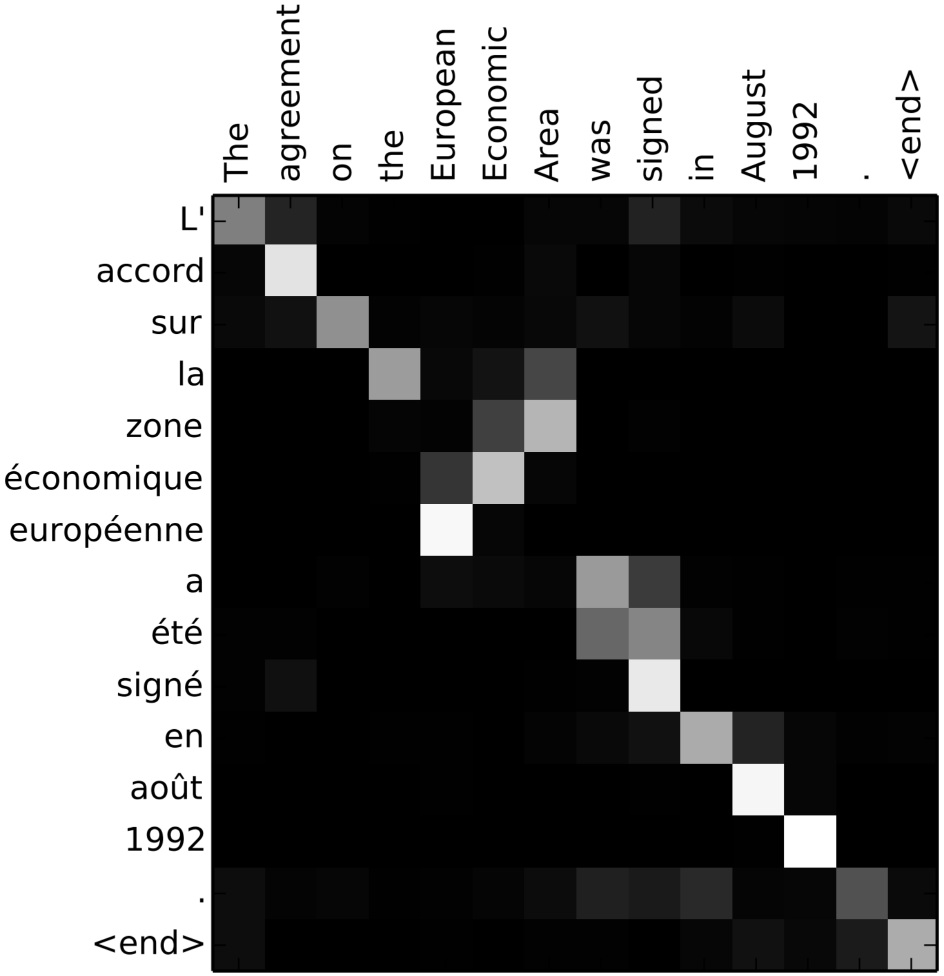
- **Input sequence not bottlenecked through single vector**
- **At each timestep of decoder, context vector "looks at" different parts of the input sequence**



Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence-to-Sequence with RNNs **and Attention**

Visualize attention weights $a_{t,i}$



Example: English to French translation

Input: “The agreement on the European Economic Area was signed in August 1992.”

Output: “L’accord sur la zone économique européenne a été signé en août 1992.”

Bahdanau et al, “Neural machine translation by jointly learning to align and translate”, ICLR 2015

Sequence-to-Sequence with RNNs **and Attention**

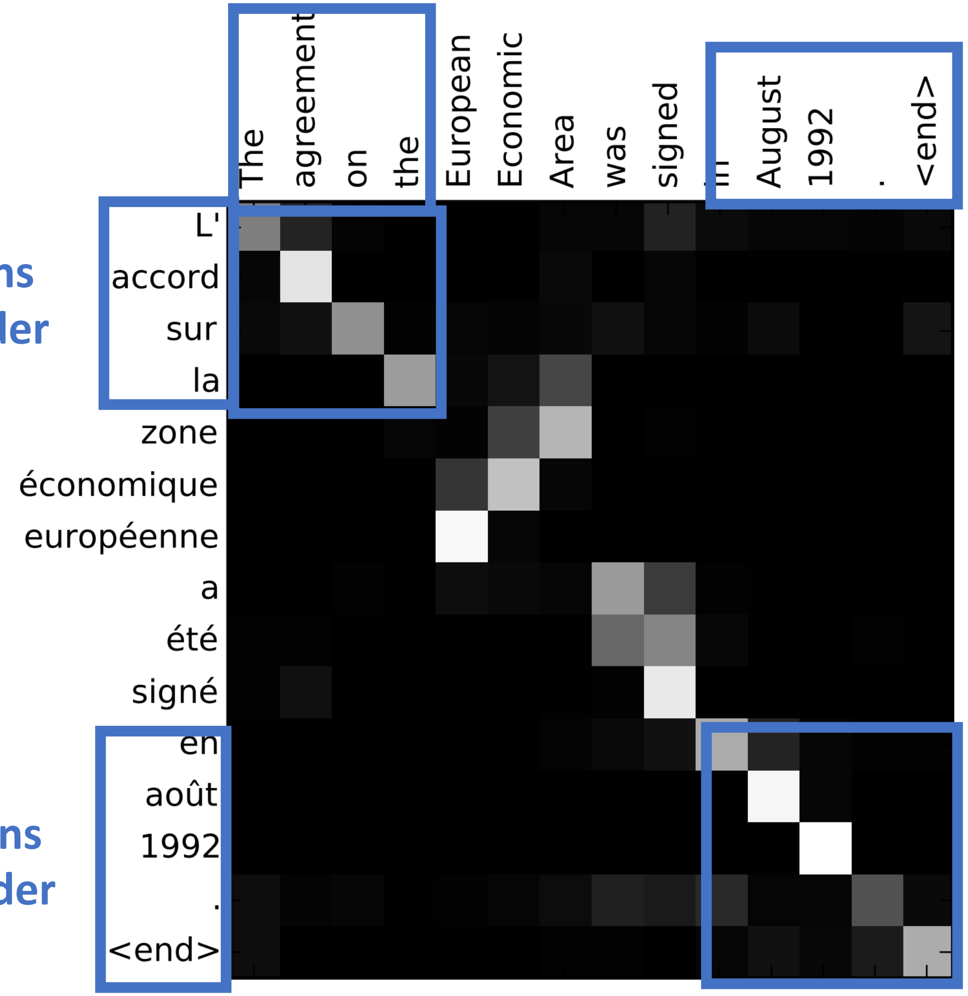
Example: English to French translation

Input: "**The agreement on the** European Economic Area was signed **in August 1992**."

Output: "**L'accord sur la** zone économique européenne a été signé **en août 1992**."

Visualize attention weights $a_{t,i}$

Diagonal attention means words correspond in order



Diagonal attention means words correspond in order

Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

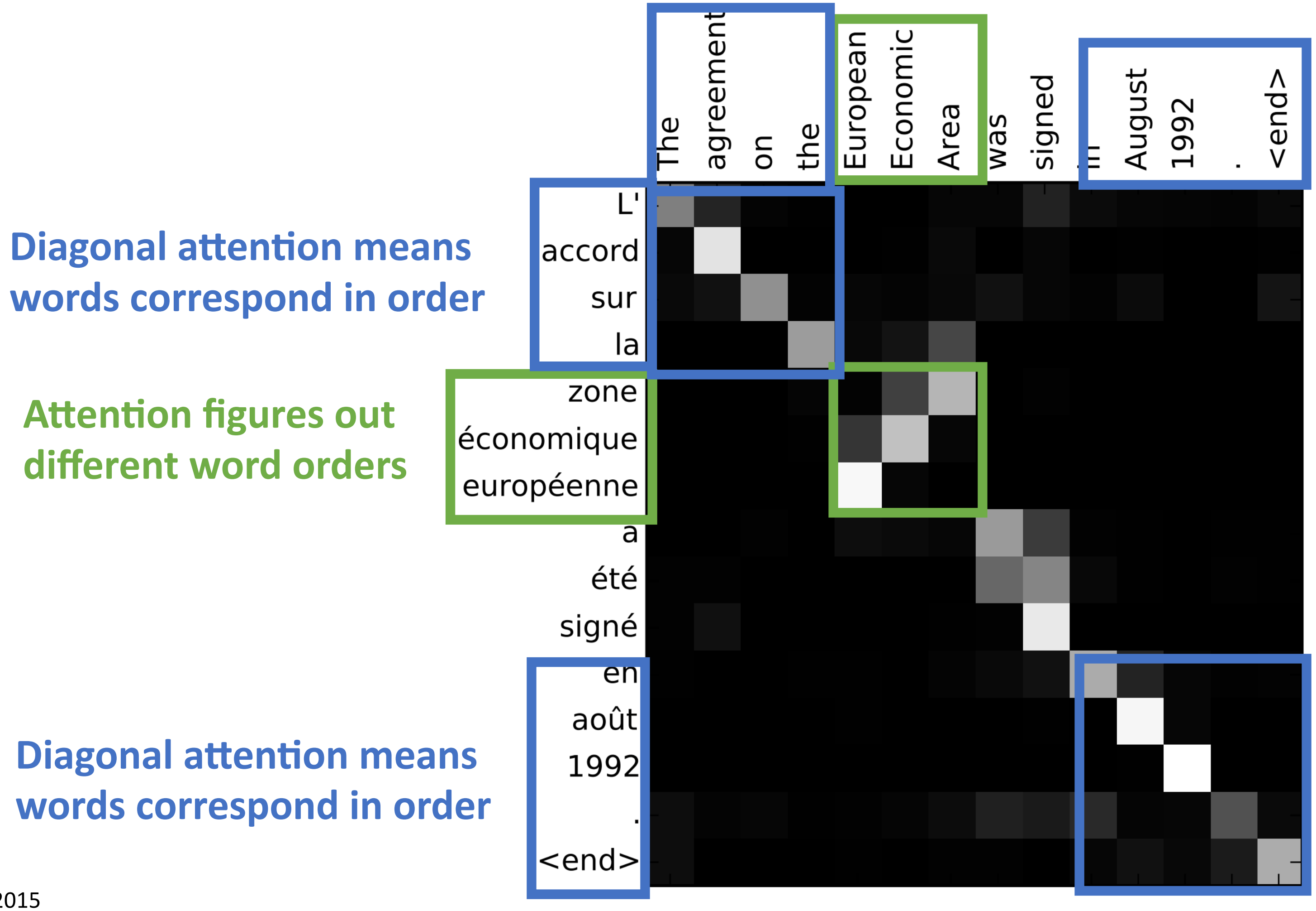
Sequence-to-Sequence with RNNs **and Attention**

Example: English to French translation

Input: "**The agreement on the European Economic Area** was signed **in August 1992**."

Output: "**L'accord sur la zone économique européenne** a été signé **en août 1992**."

Visualize attention weights $a_{t,i}$



Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence-to-Sequence with RNNs **and Attention**

Example: English to French translation

Input: "**The agreement on the European Economic Area was signed in August 1992**."

Output: "**L'accord sur la zone économique européenne a été signé en août 1992**."

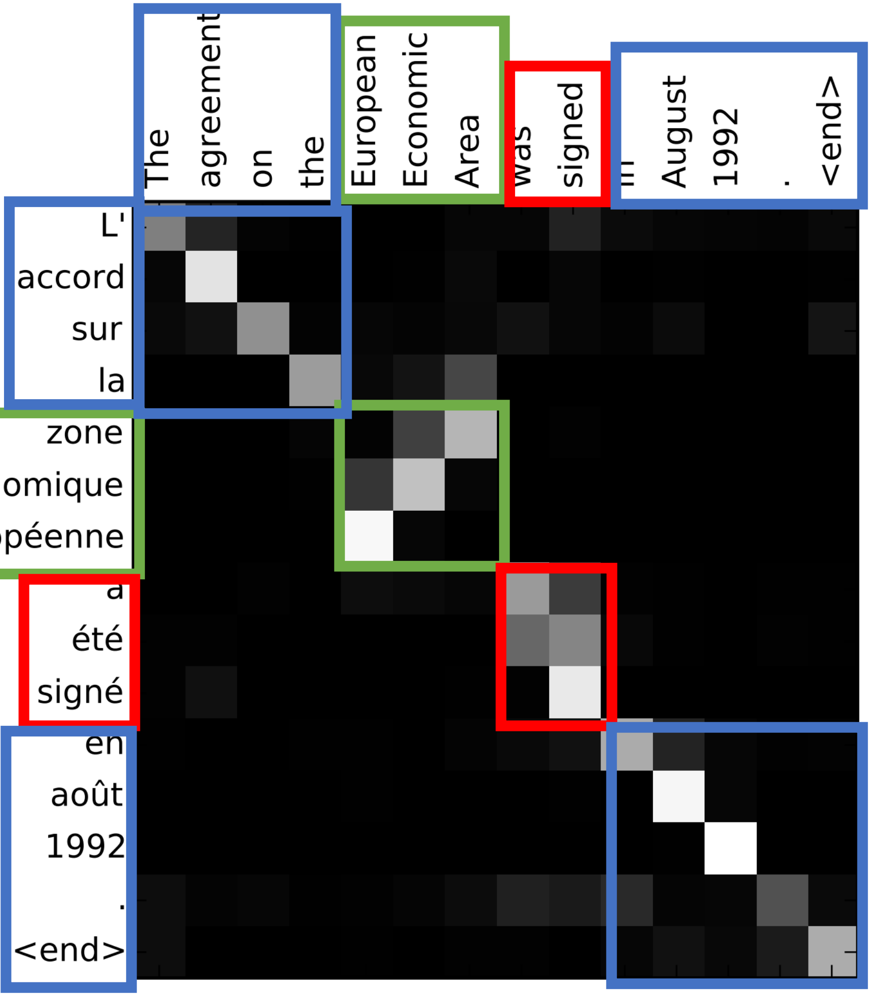
Visualize attention weights $a_{t,i}$

Diagonal attention means words correspond in order

Attention figures out different word orders

Verb conjugation

Diagonal attention means words correspond in order

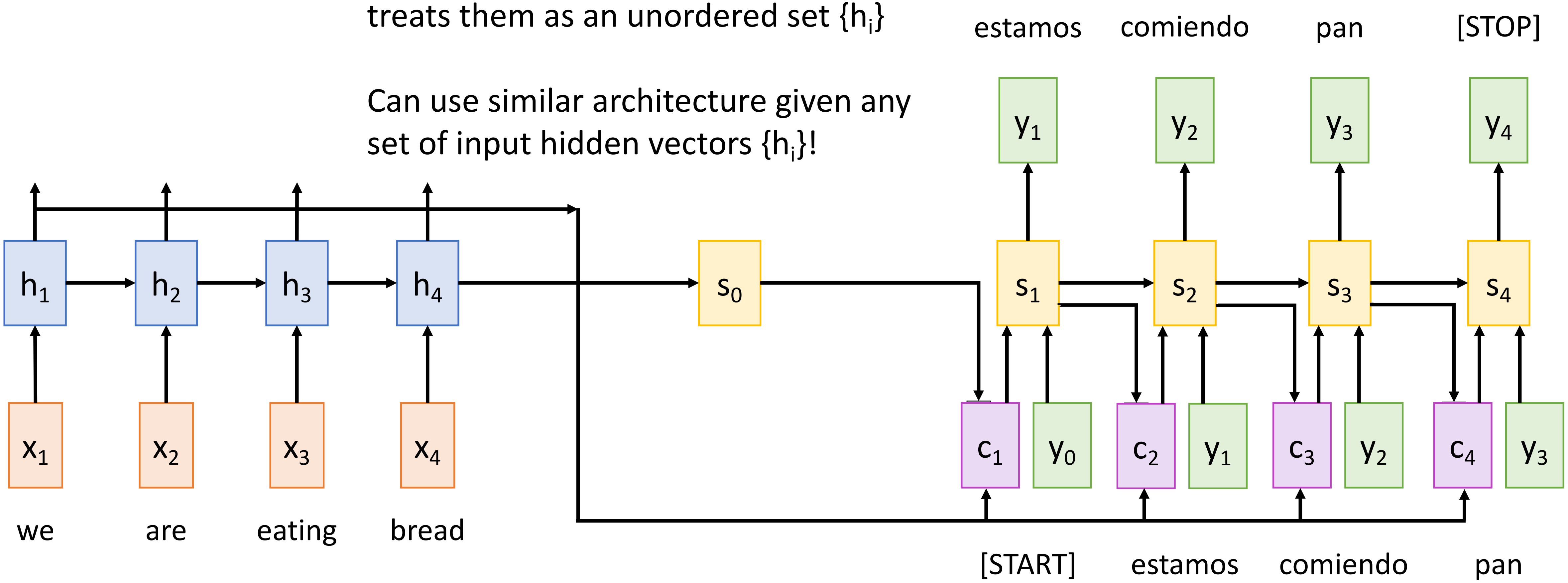


Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence-to-Sequence with RNNs **and Attention**

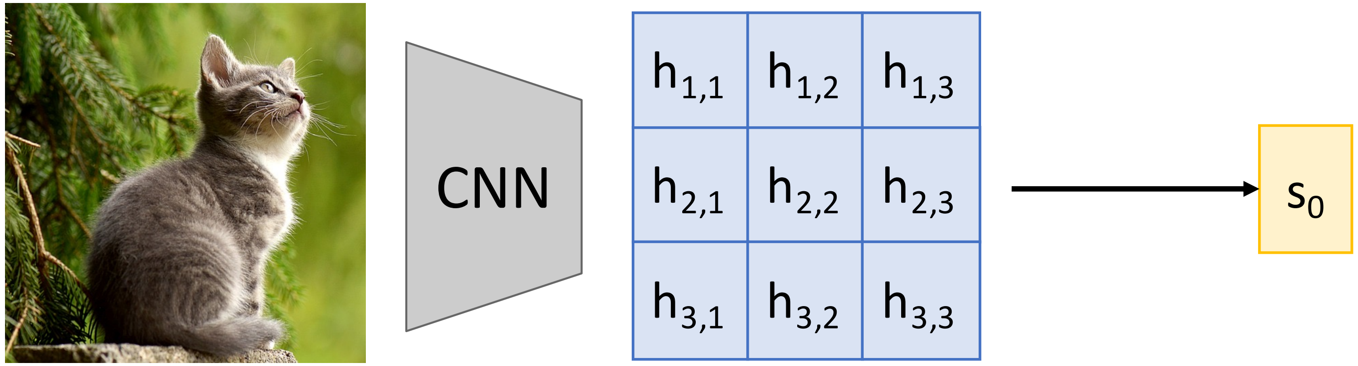
The decoder doesn't use the fact that h_i form an ordered sequence – it just treats them as an unordered set $\{h_i\}$

Can use similar architecture given any set of input hidden vectors $\{h_i\}$!



Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Image Captioning with RNNs and Attention

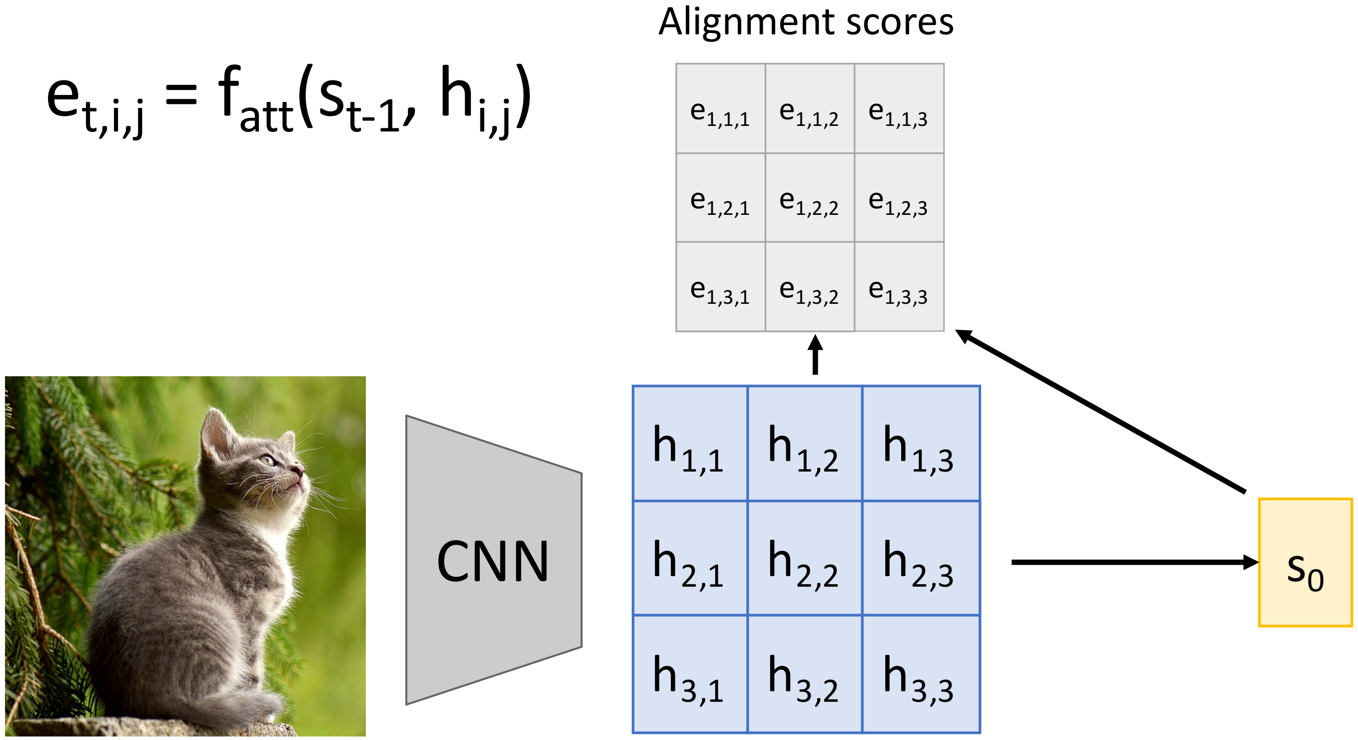


Use a CNN to compute a
grid of features for an image

Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with RNNs and Attention

$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$

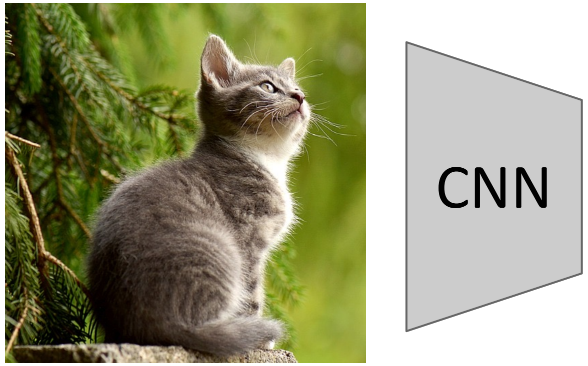


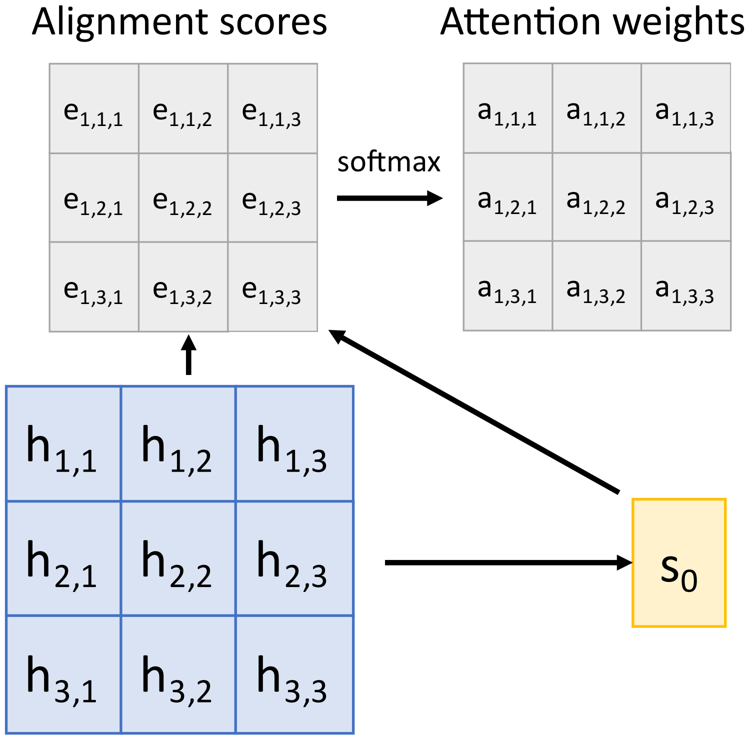
Use a CNN to compute a
grid of features for an image

Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with RNNs and Attention

$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$
$a_{t,:,:} = \text{softmax}(e_{t,:,:})$



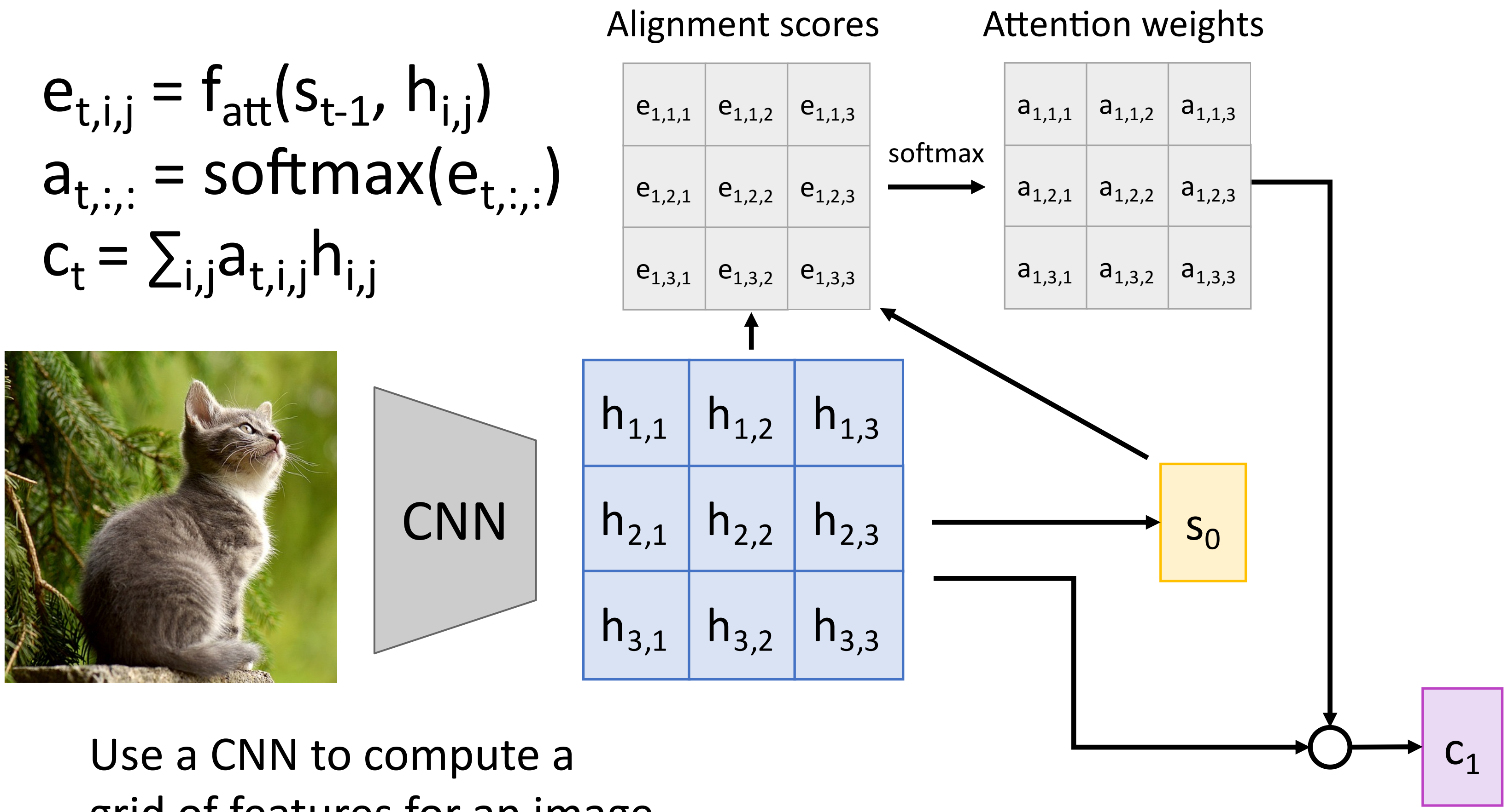


Use a CNN to compute a
grid of features for an image

Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with RNNs and Attention

$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$
$a_{t,:,:} = \text{softmax}(e_{t,:,:})$
$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$

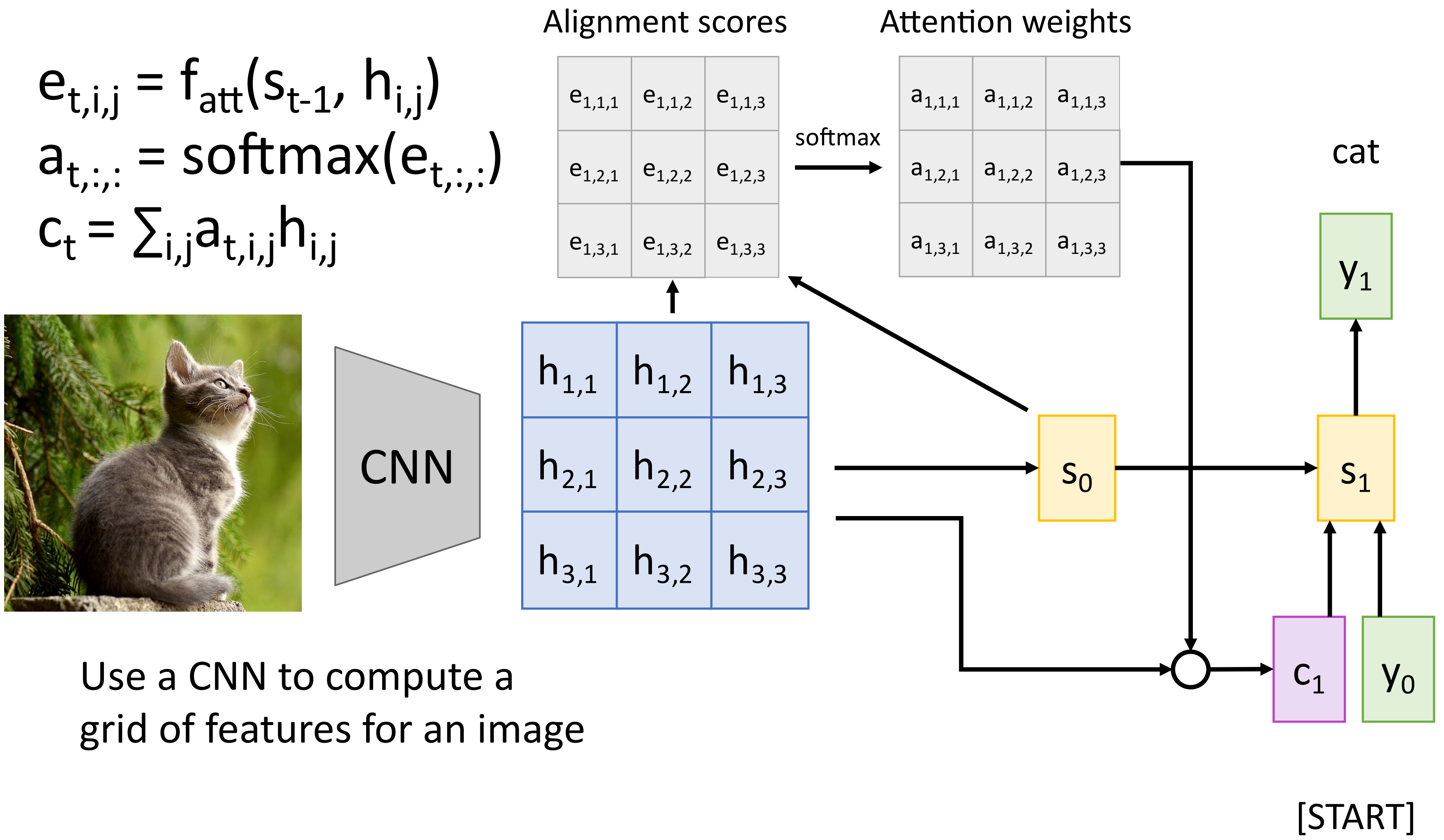


Use a CNN to compute a
grid of features for an image

Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with RNNs and Attention

$$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$$
$$a_{t,:,:} = \mathrm{softmax}(e_{t,:,:})$$
$$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$$



Use a CNN to compute a grid of features for an image

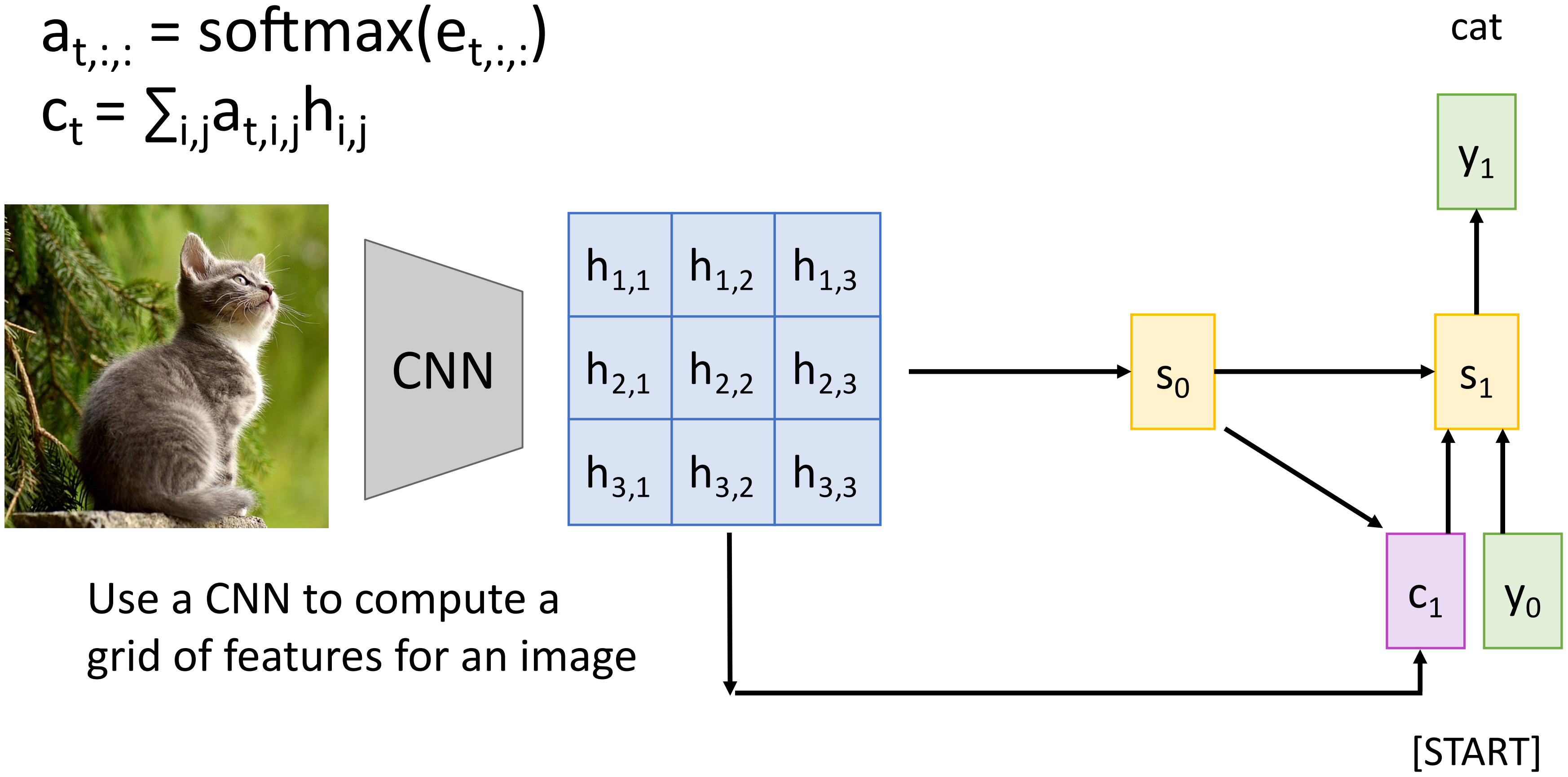
Xu et al, “Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Image Captioning with RNNs and Attention

$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$

$a_{t,:,:} = \text{softmax}(e_{t,:,:})$

$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$

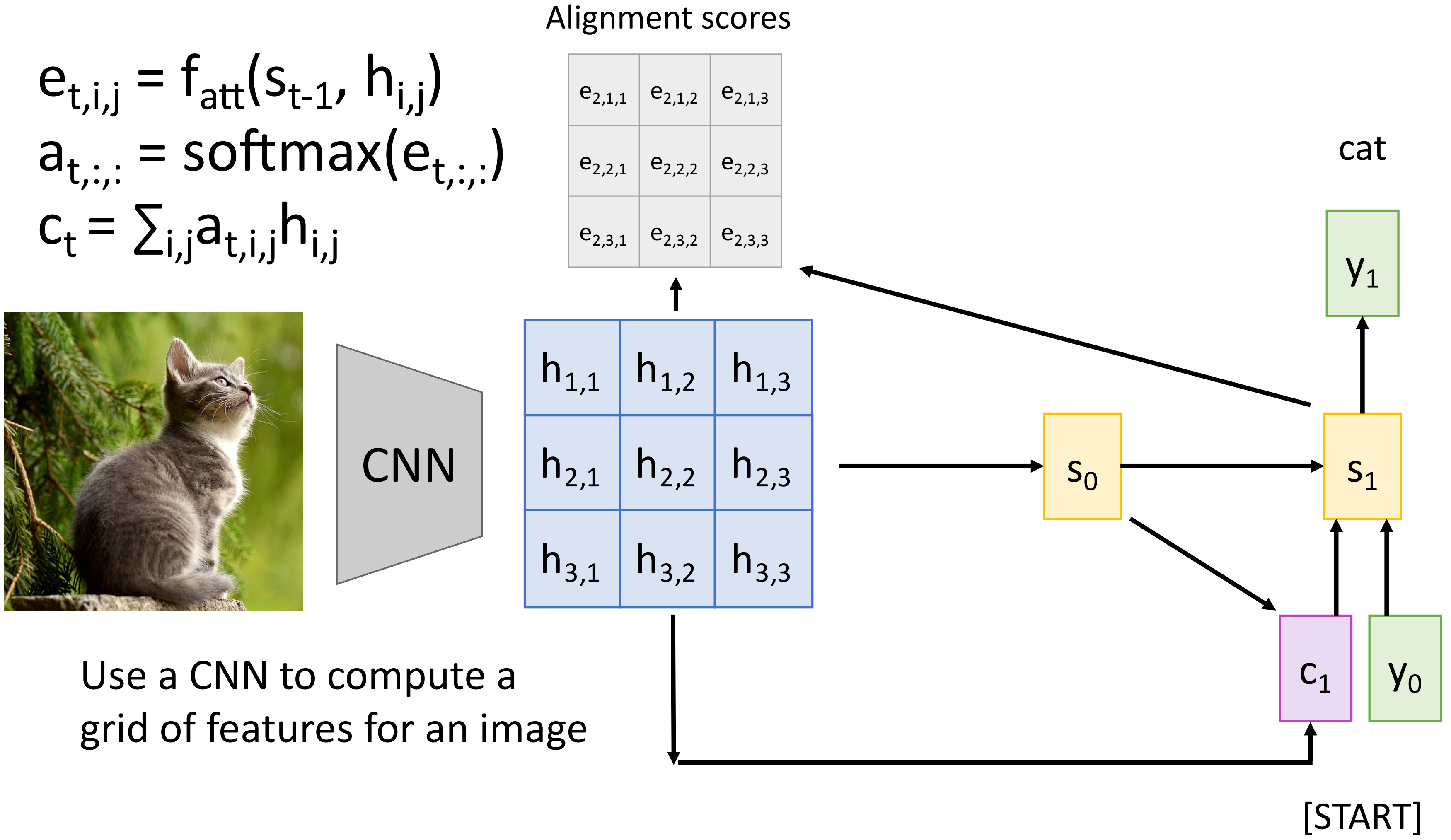


Use a CNN to compute a grid of features for an image

Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with RNNs and Attention

$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$
$a_{t,:,:} = \text{softmax}(e_{t,:,:})$
$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$

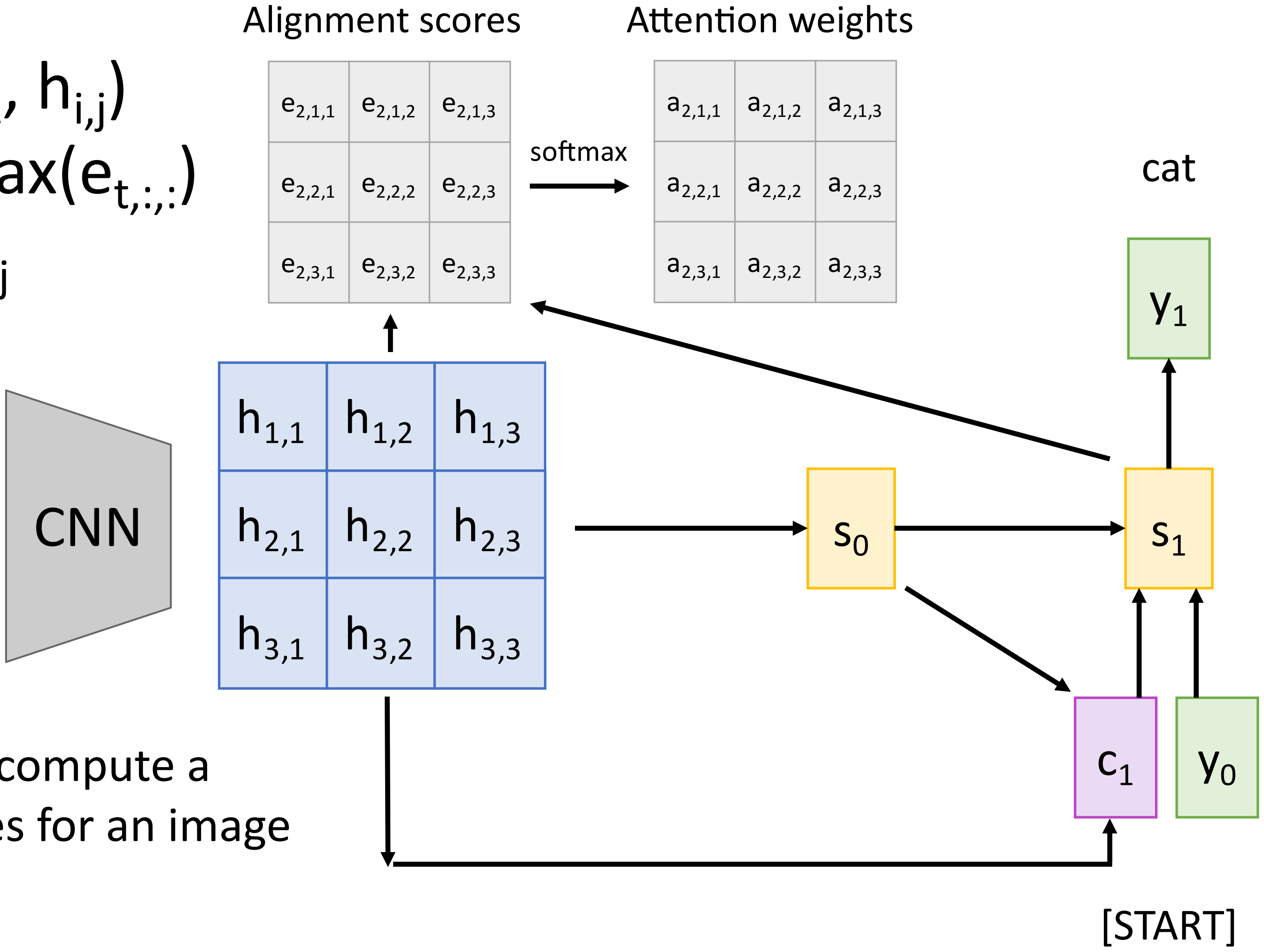


Use a CNN to compute a grid of features for an image

Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with RNNs and Attention

$$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$$
$$a_{t,:,:} = \text{softmax}(e_{t,:,:})$$
$$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$$

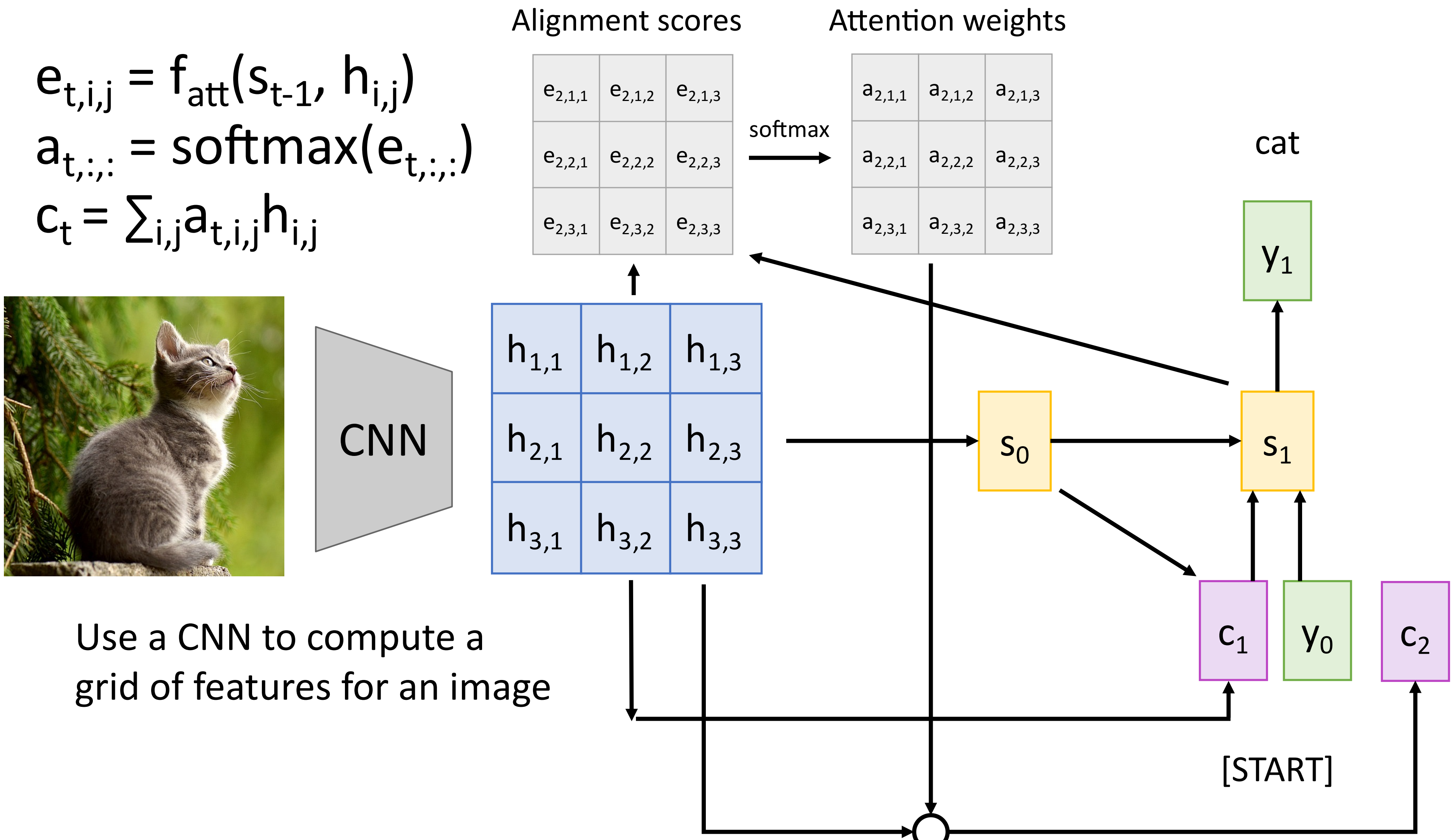


Use a CNN to compute a grid of features for an image

Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with RNNs and Attention

$$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$$
$$a_{t,:,:} = \mathrm{softmax}(e_{t,:,:})$$
$$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$$



Use a CNN to compute a
grid of features for an image

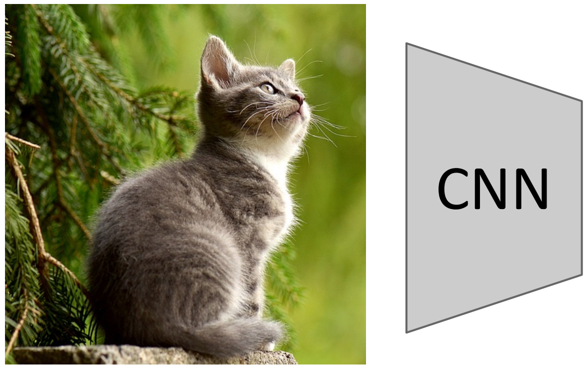
Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

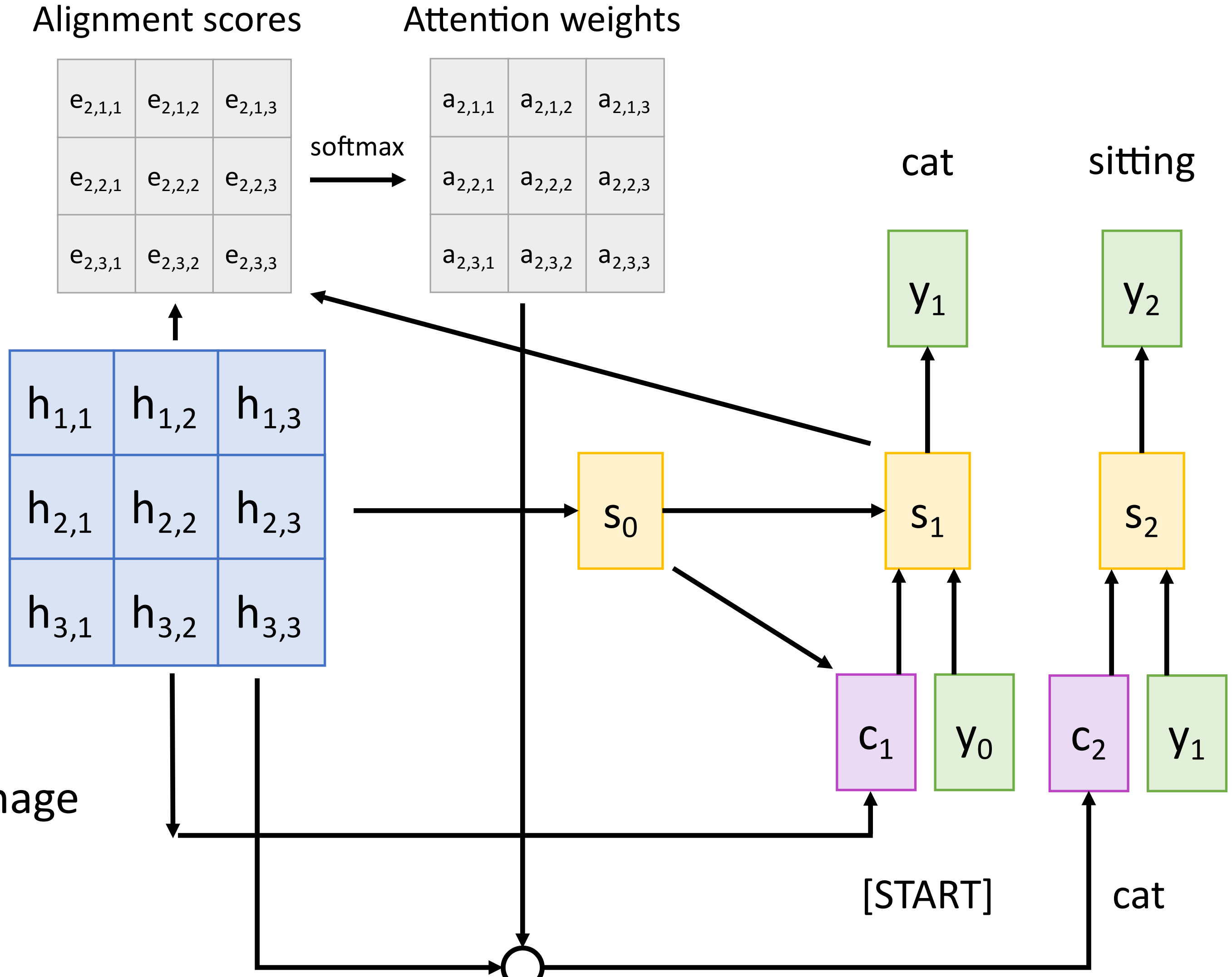
Image Captioning with RNNs and Attention

$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$

$a_{t,:,:} = \text{softmax}(e_{t,:,:})$

$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$





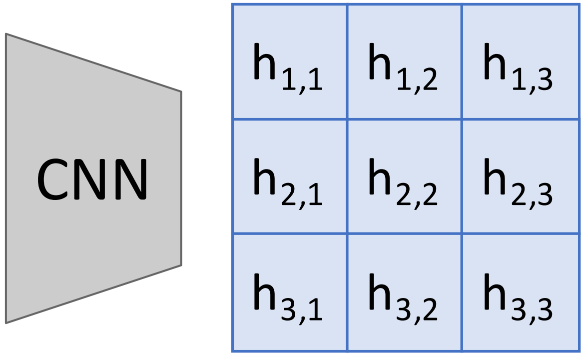
Use a CNN to compute a grid of features for an image

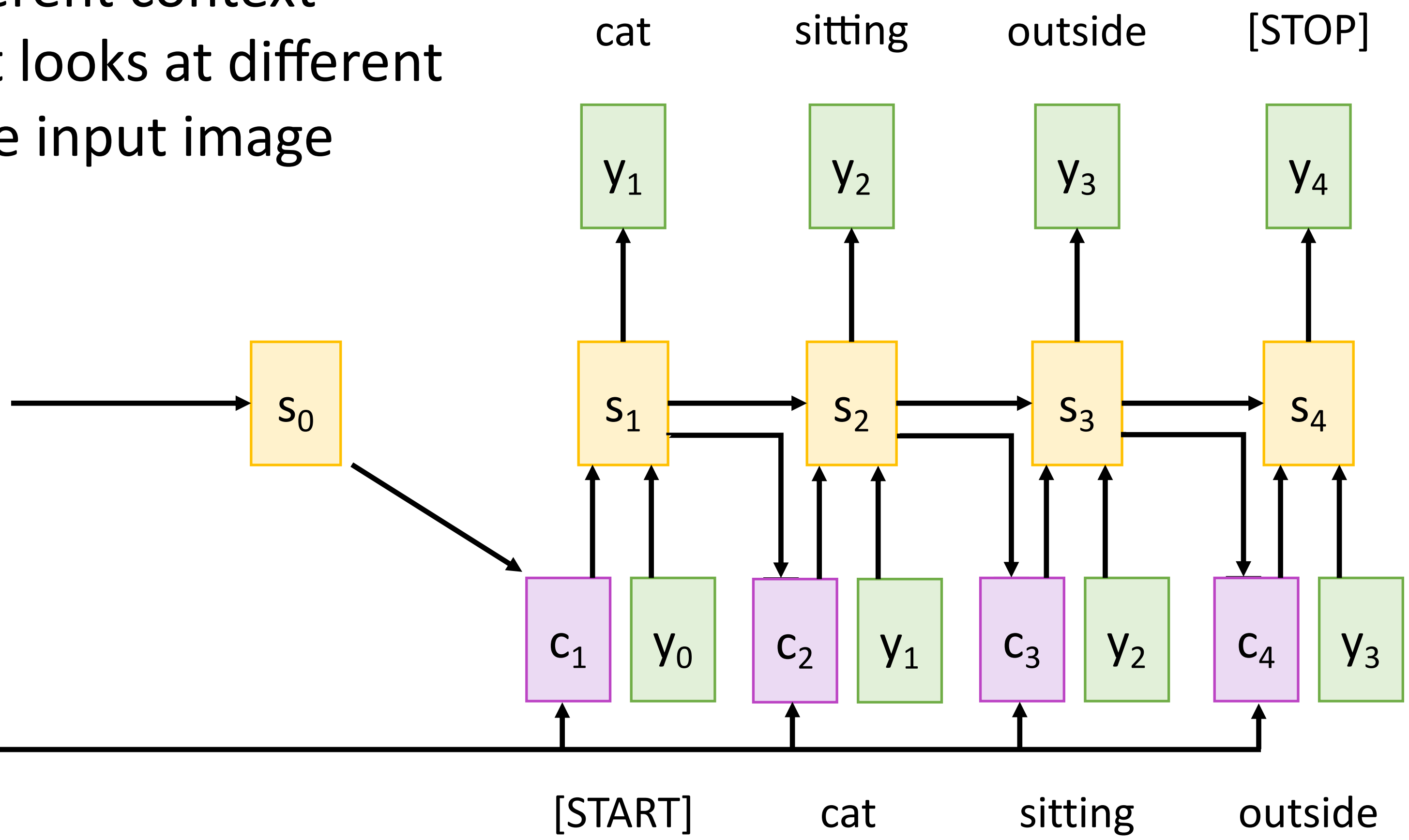
Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with RNNs and Attention

$$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$$
$$a_{t,:,:} = \mathrm{softmax}(e_{t,:,:})$$
$$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$$

Each timestep of decoder uses a different context vector that looks at different parts of the input image

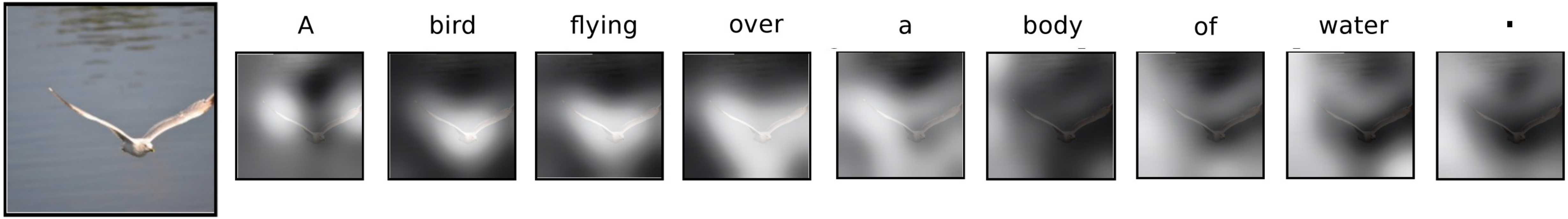




Use a CNN to compute a grid of features for an image

Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with RNNs and Attention



Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with RNNs and Attention

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A group of people sitting on a boat in the water.

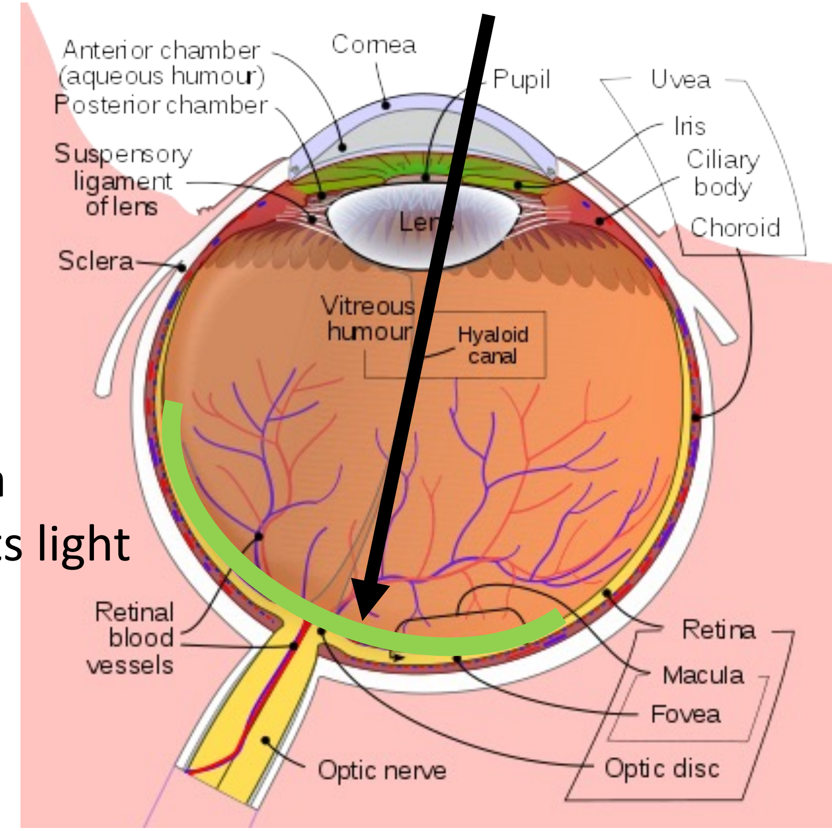
A giraffe standing in a forest with trees in the background.

Xu et al, “Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Human Vision: Fovea 视网膜的中央凹

Light enters eye

Retina
detects light

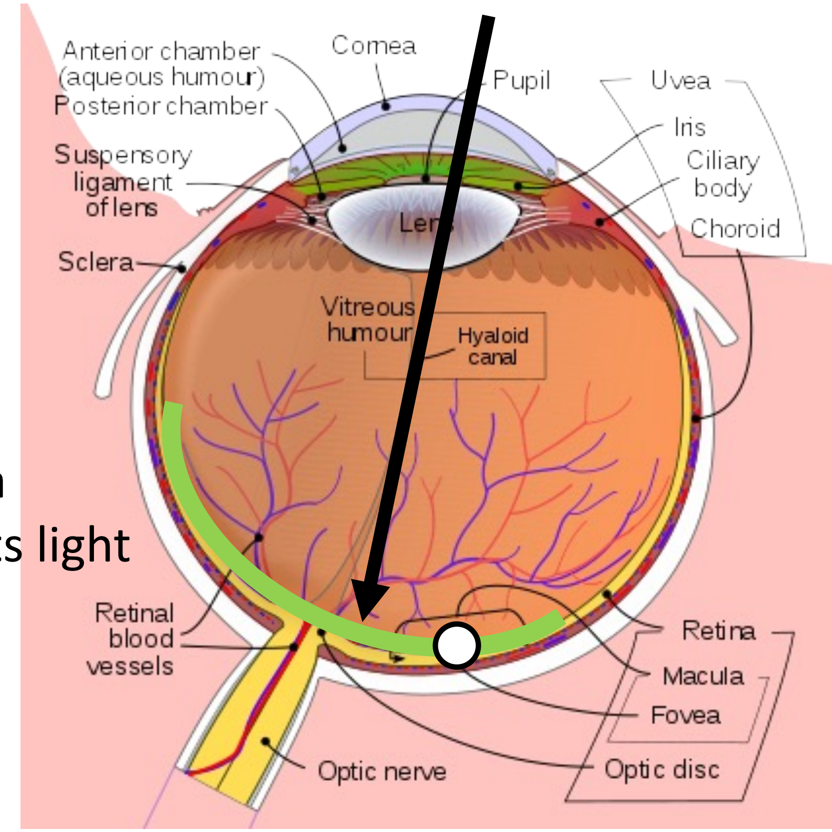


Human Vision: Fovea 视网膜的中央凹

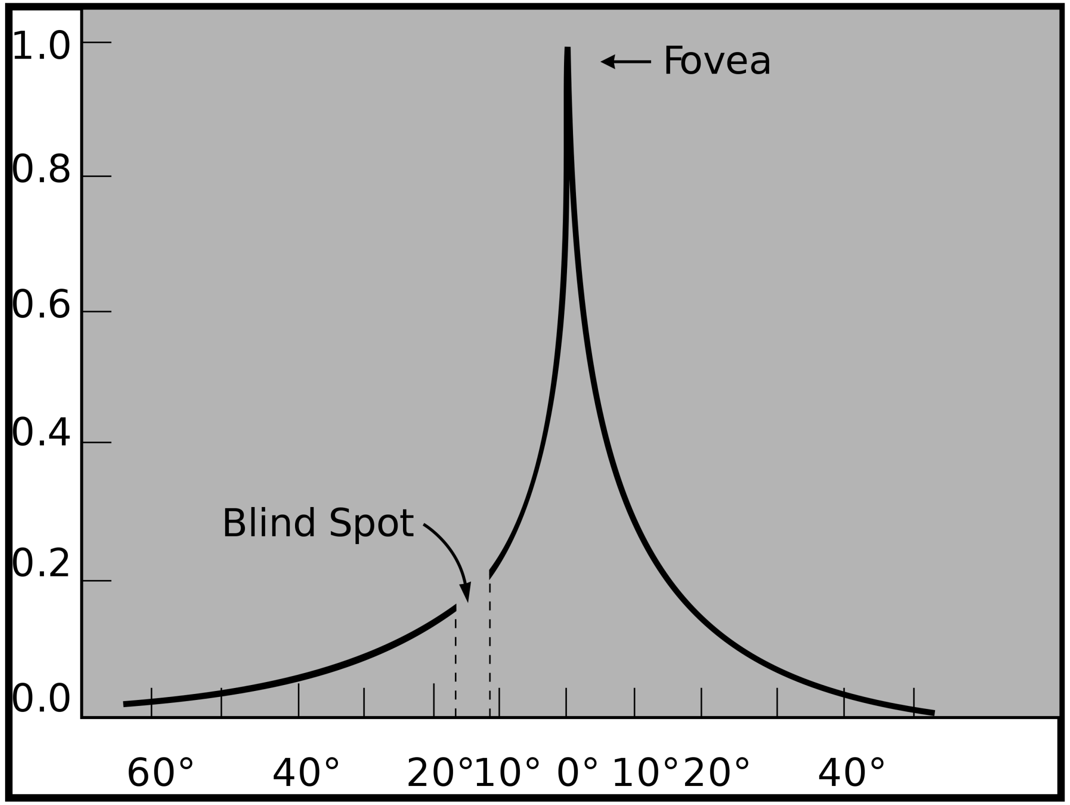
Light enters eye

视网膜

Retina
detects light

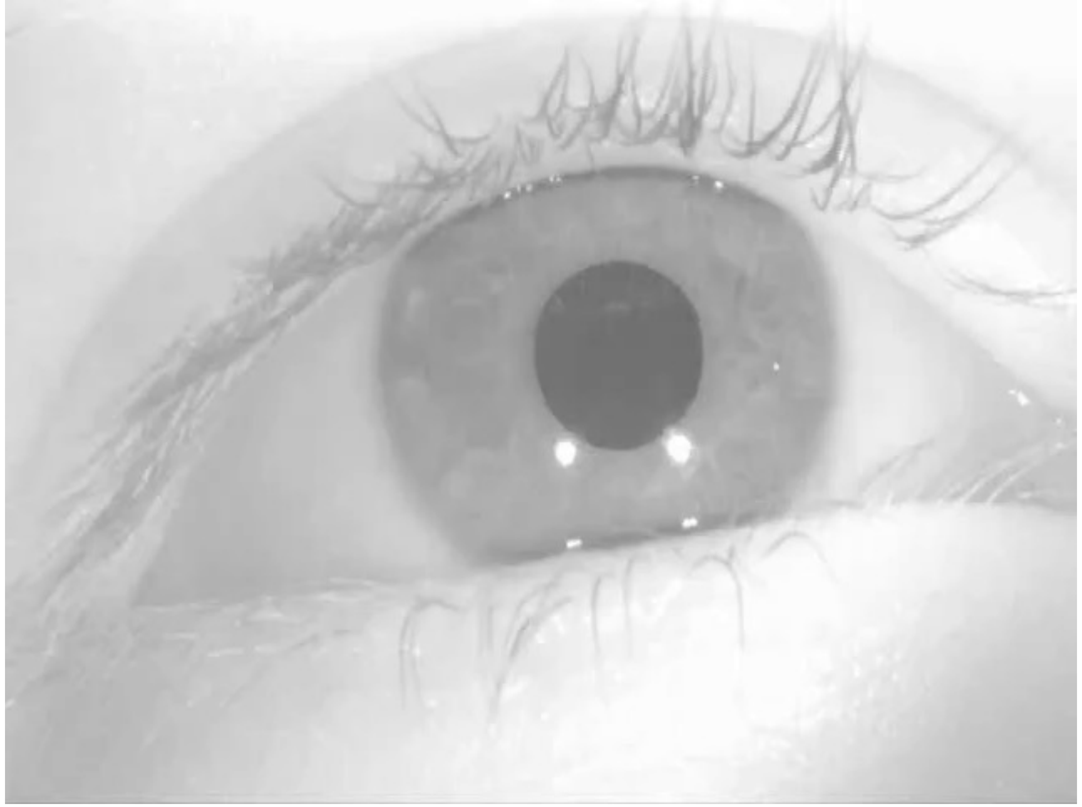


The **fovea** is a tiny region of the retina that can see with high acuity



Human Vision: Saccades 扫视

Human eyes are constantly moving so we don't notice



The **fovea** is a tiny region of the retina that can see with high acuity 敏度

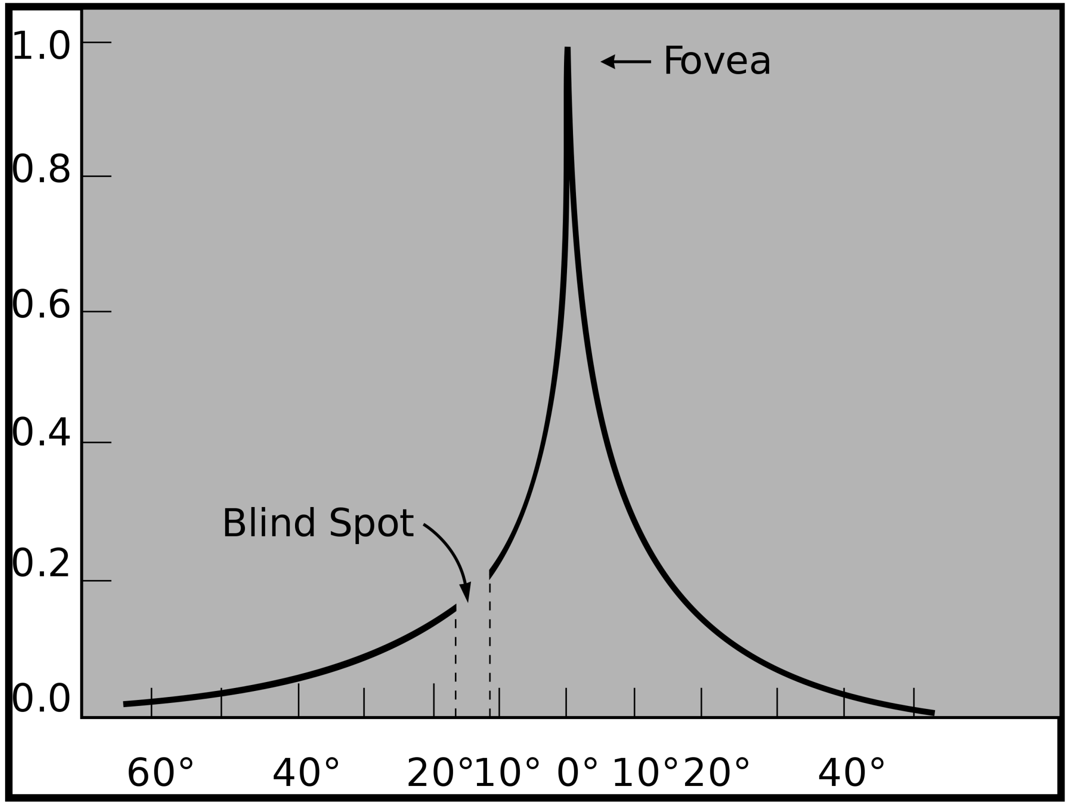
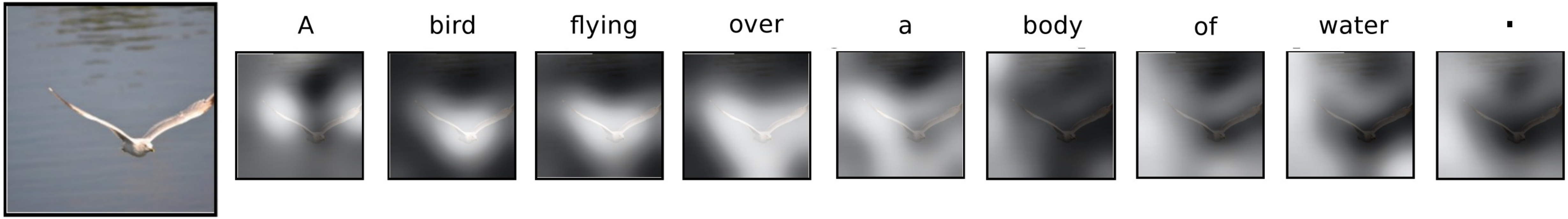
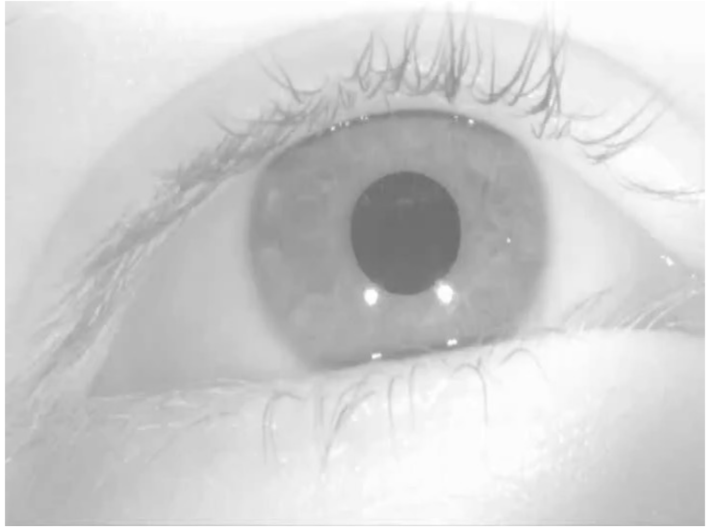


Image Captioning with RNNs and Attention



Attention weights at each timestep kind of like saccades of human eye



Xu et al, “Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

X, Attend, and Y

"**Show, attend, and tell**" *(Xu et al, ICML 2015)*
Look at image, attend to image regions, produce question

"**Ask, attend, and answer**" *(Xu and Saenko, ECCV 2016)*
"**Show, ask, attend, and answer"** *(Kazemi and Elqursh, 2017)*
Read text of question, attend to image regions, produce answer

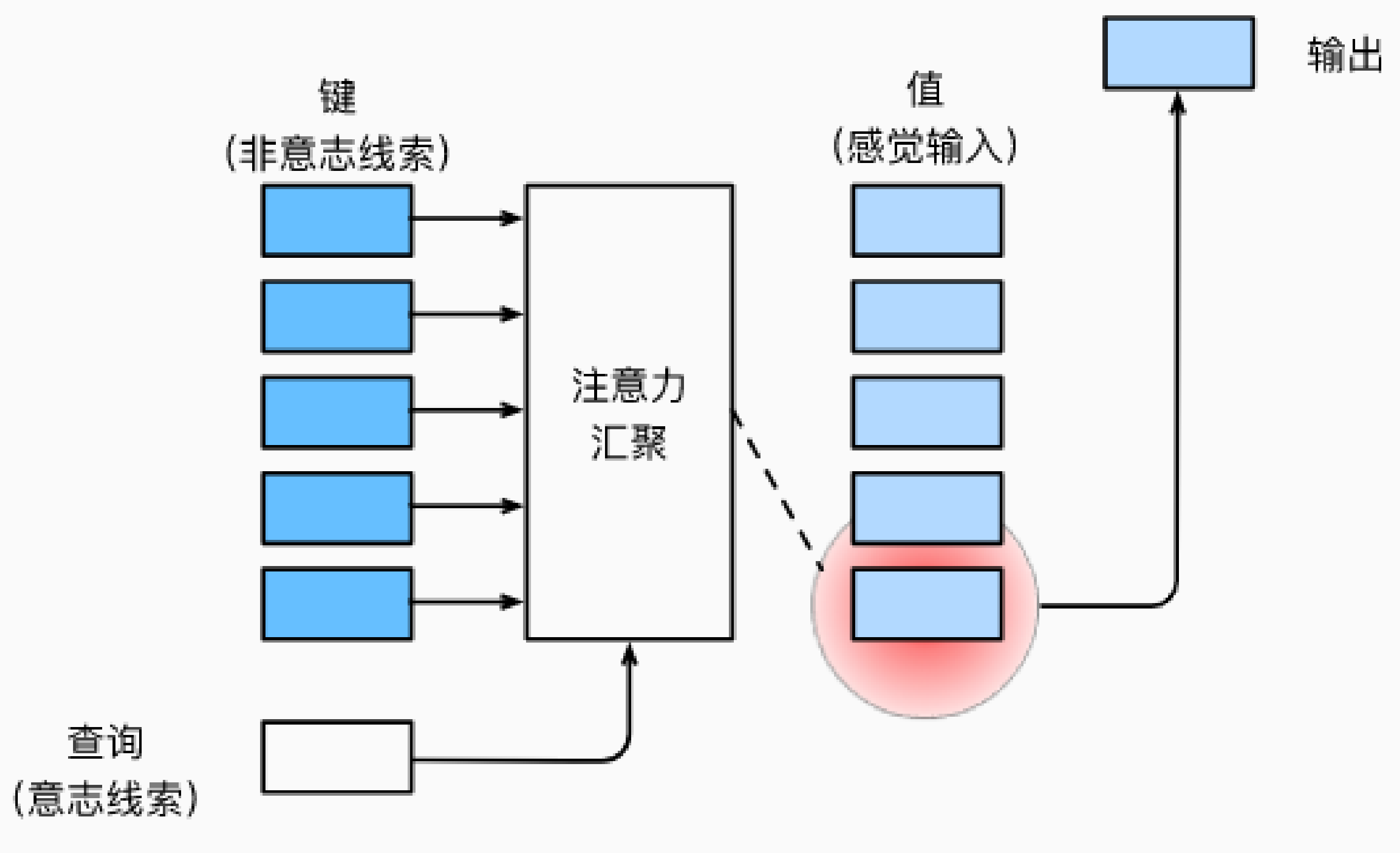
"**Listen, attend, and spell**" *(Chan et al, ICASSP 2016)*
Process raw audio, attend to audio regions while producing text

"**Listen, attend, and walk**" *(Mei et al, AAAI 2016)*
Process text, attend to text regions, output navigation commands

"**Show, attend, and interact**" *(Qureshi et al, ICRA 2017)*
Process image, attend to image regions, output robot control commands

"**Show, attend, and read**" *(Li et al, AAAI 2019)*
Process image, attend to image regions, output text

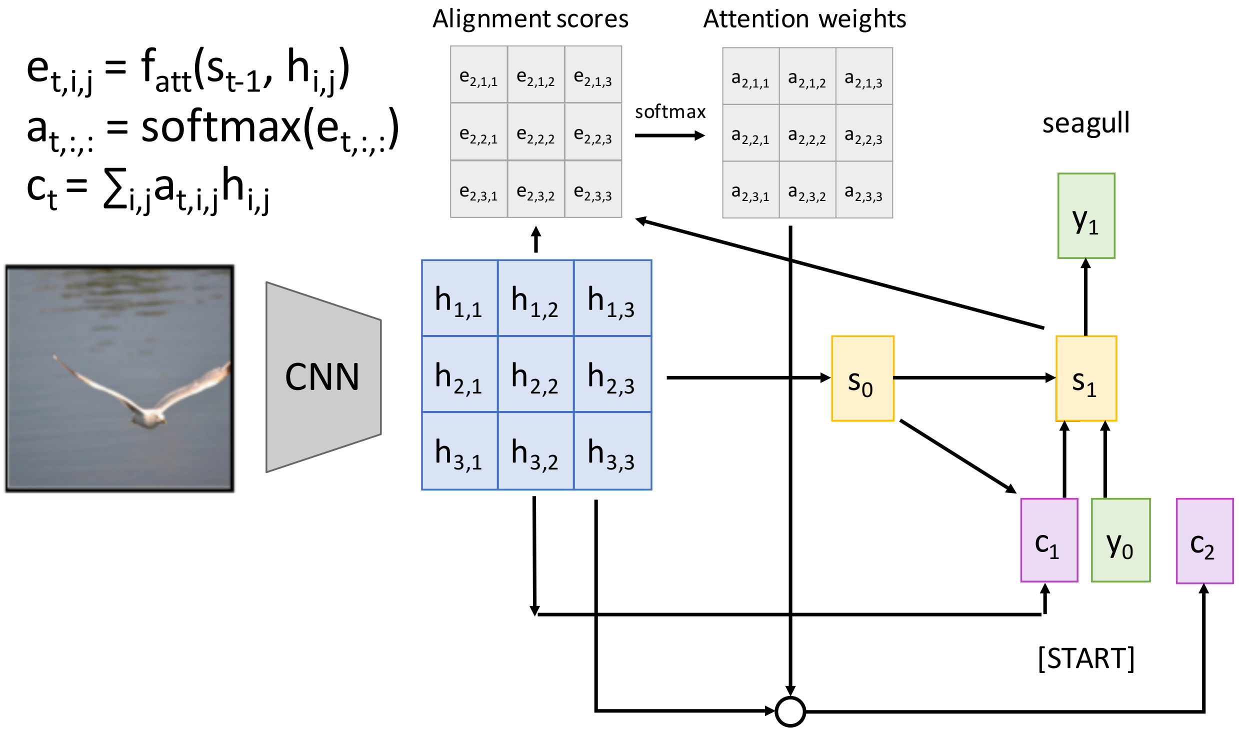
自主性与非自主性注意力



“ 是否包含自主性提示 ” 将注意力机制与全连接层或汇聚层区别开来。在注意力机制的背景下，自主性提示被称为查询（query）。 给定任何查询，注意力机制通过注意力汇聚（attention pooling） 将选择引导至感官输入，即值（value）。每个值都与一个键（key）配对， 可以想象为感官输入的非自主提示。

Attention Layer

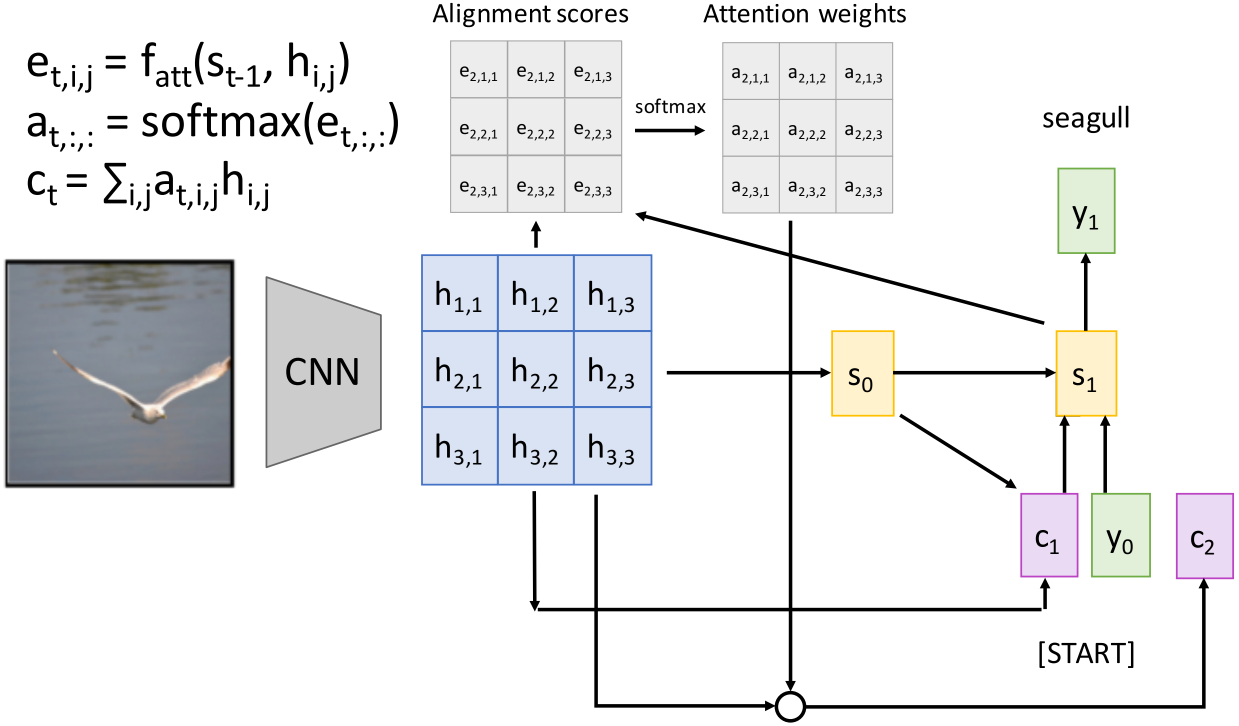
Inputs:
Query vector: $\mathbf{q}$ (Shape: D_Q)
Input vectors: $\mathbf{X}$ (Shape: N_X x D_X)
Similarity function: f_{att}



Computation:
Similarities: e (Shape: N_X) $e_i = f_{att}(\mathbf{q}, \mathbf{X}_i)$
Attention weights: a = softmax(e) (Shape: N_X)
Output vector: $y = \sum_i a_i \mathbf{X}_i$ (Shape: D_X)

Attention Layer

$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$
$a_{t,:,:} = \text{softmax}(e_{t,:,:})$
$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$



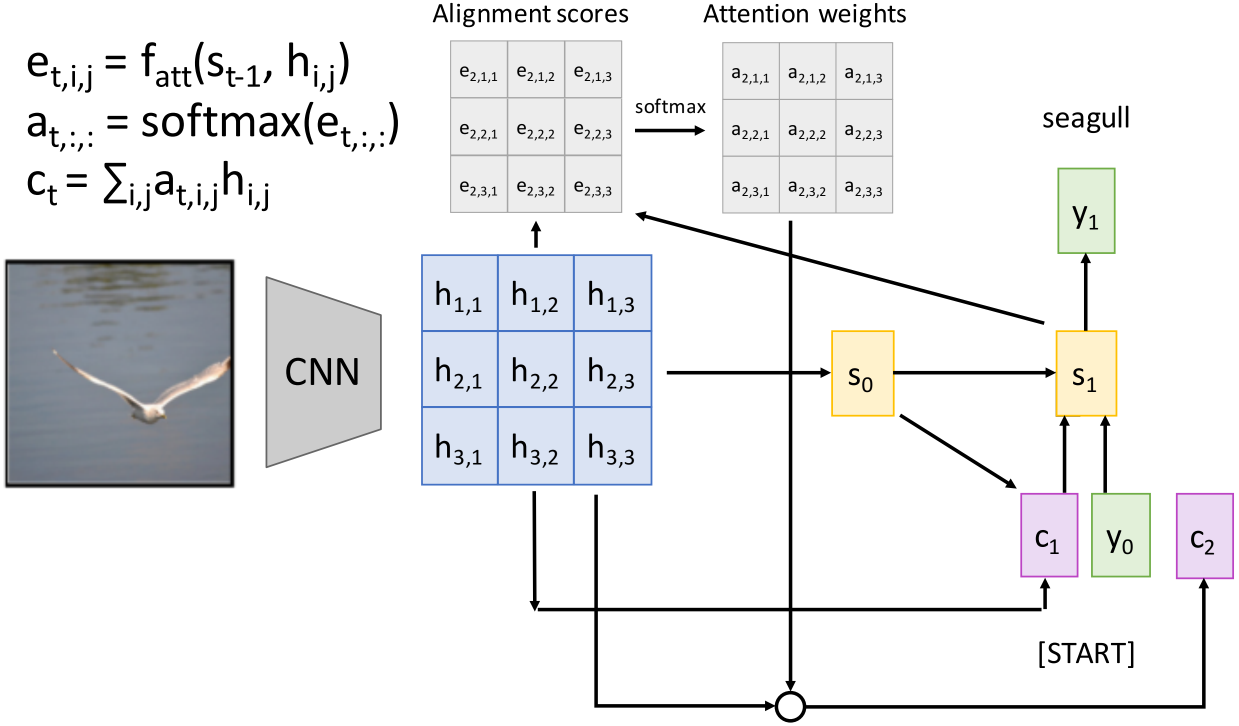
Inputs:
Query vector: $\mathbf{q}$ (Shape: D_Q)
Input vectors: $\mathbf{X}$ (Shape: N_X x D_Q)
Similarity function: dot product

Computation:
Similarities: e (Shape: N_X) $e_i = \mathbf{q} \cdot \mathbf{X}_i$
Attention weights: a = softmax(e) (Shape: N_X)
Output vector: $y = \sum_i a_i \mathbf{X}_i$ (Shape: D_X)

Changes:
- Use dot product for similarity

Attention Layer

$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$
$a_{t,:,:} = \text{softmax}(e_{t,:,:})$
$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$



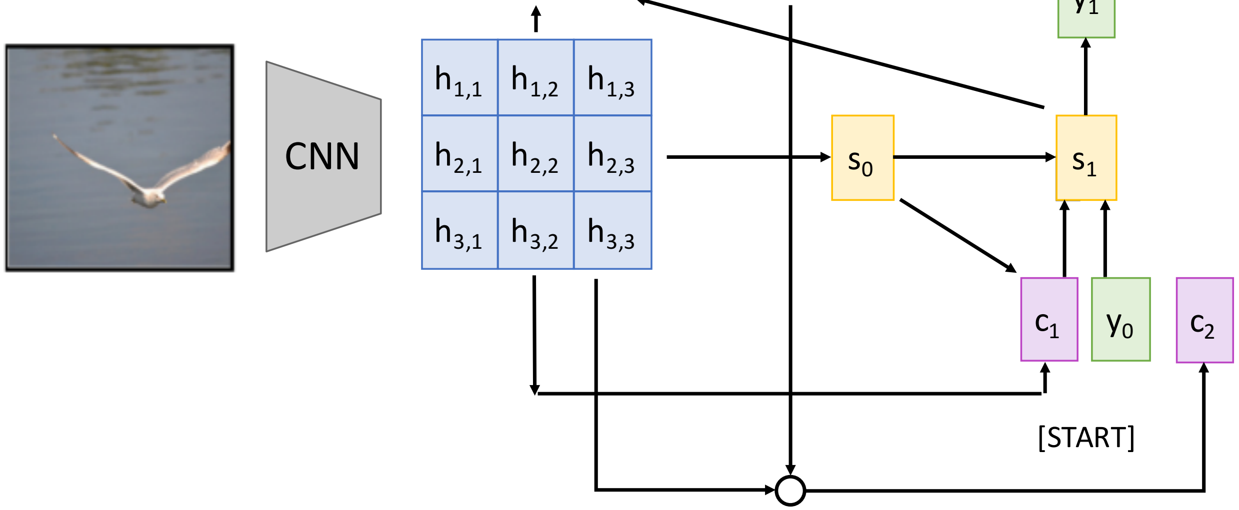
Inputs:
Query vector: $\mathbf{q}$ (Shape: D_Q)
Input vectors: $\mathbf{X}$ (Shape: N_X x D_Q)
Similarity function: scaled dot product

Computation:
Similarities: e (Shape: N_X) $e_i = \mathbf{q} \cdot \mathbf{X}_i \boxed{/ \sqrt{D_Q}}$
Attention weights: a = softmax(e) (Shape: N_X)
Output vector: $y = \sum_i a_i \mathbf{X}_i$ (Shape: D_X)

Changes:
- Use **scaled** dot product for similarity

Attention Layer

$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$
$a_{t,:,:} = \text{softmax}(e_{t,:,:})$
$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$



Inputs:

Query vector: $\mathbf{q}$ (Shape: D_Q)

Input vectors: $\mathbf{X}$ (Shape: N_X x D_Q)

Similarity function: $\boxed{\text{scaled dot product}}$

Large similarities will cause softmax to saturate and give vanishing gradients

Recall $a \cdot b = |a||b| \cos(\text{angle})$

Suppose that a and b are constant vectors of dimension D

Then $|a| = (\sum_i a^2)^{1/2} = a\sqrt{D}$

Computation:

Similarities: e (Shape: N_X) $e_i = \mathbf{q} \cdot \mathbf{X}_i \boxed{/ \sqrt{D_Q}}$

Attention weights: $a = \text{softmax}(e)$ (Shape: N_X)

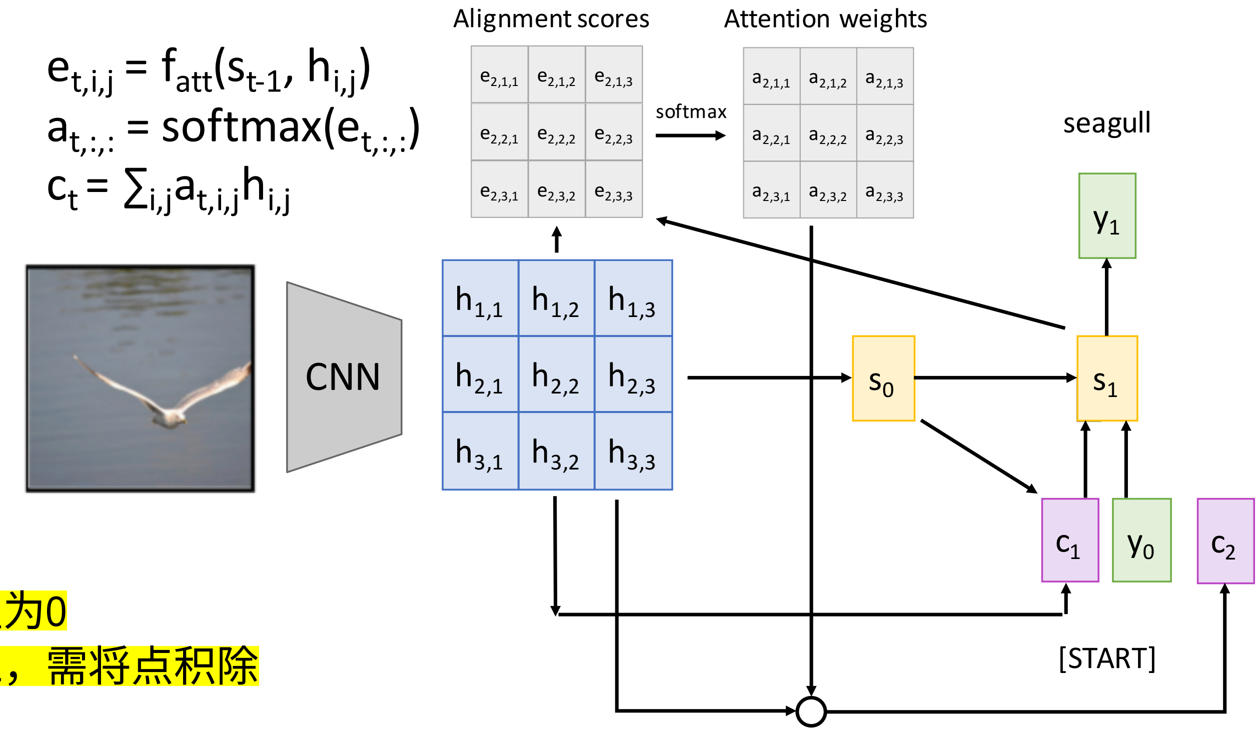
Output vector: $y = \sum_i a_i \mathbf{X}_i$ (Shape: D_X)

Changes:

- Use **scaled** dot product for similarity

Attention Layer

$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$
$a_{t,:,:} = \text{softmax}(e_{t,:,:})$
$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$



Inputs:
Query vectors: $\boxed{\mathbf{Q}\ (\text{Shape: } N_Q \times D_Q)}$
Input vectors: $\mathbf{X}$ (Shape: $N_X \times D_Q$)

假设查询和键的所有元素都是独立的随机变量，
并且都满足零均值和单位方差， 那么两个向量的点积的均值为0
，方差为d。 为确保无论向量长度如何，点积的方差仍然是1，需将点积除
以sqrt(D_Q)，称之为缩放点积注意力。

Computation:
Similarities: $E = \mathbf{Q}\mathbf{X}^T / \sqrt{D_Q}$ (Shape: $N_Q \times N_X$) $E_{i,j} = (\mathbf{Q}_i \cdot \mathbf{X}_j) / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_Q \times N_X$)
Output vectors: $Y = A\mathbf{X}$ (Shape: $N_Q \times D_X$) $Y_i = \sum_j A_{i,j} X_j$

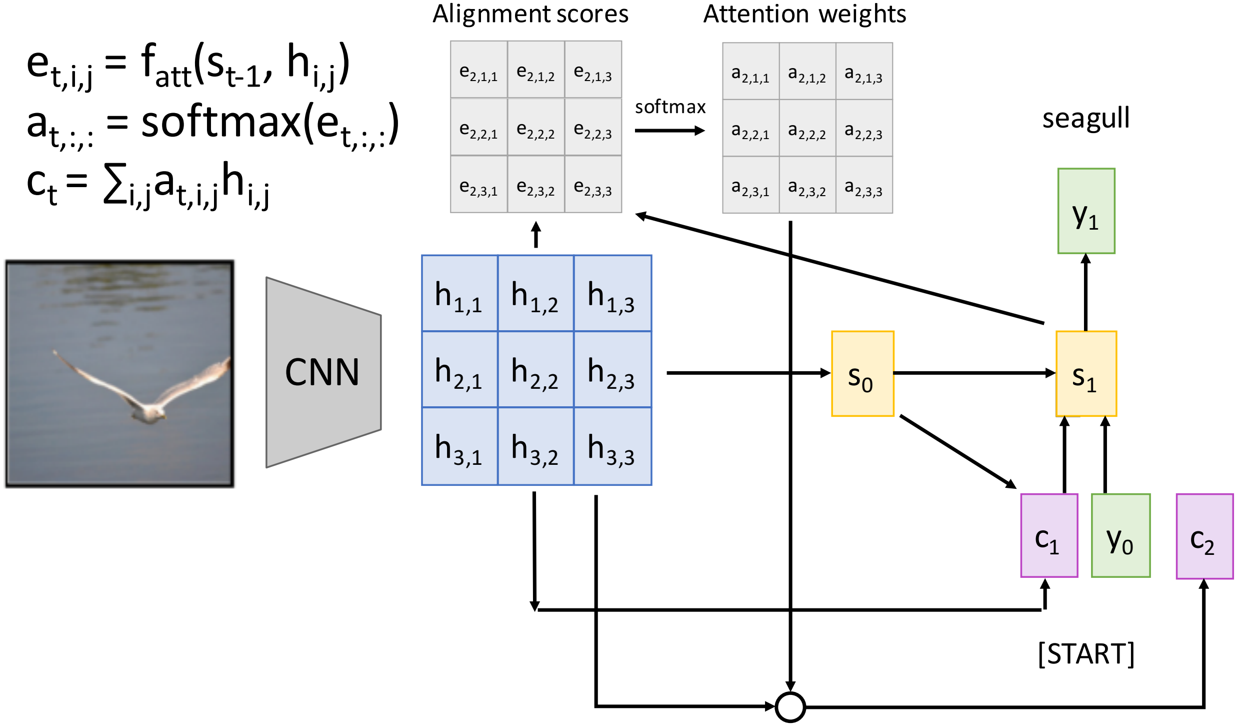
Changes:
- Use scaled dot product for similarity
- Multiple **query** vectors

Attention Layer

<u>**Inputs**</u>:
Query vectors: **Q** (Shape: N_Q x D_Q)
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)

<u>**Computation**</u>:
Key vectors: **K** = $\mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: **V** = $\mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: E = $\mathbf{QK}^T / \sqrt{D_Q}$ (Shape: N_Q x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_Q x N_X)
Output vectors: Y = A**V** (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j} \mathbf{V_j}$

$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$
$a_{t,:,:} = \text{softmax}(e_{t,:,:})$
$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$



Changes:
- Use scaled dot product for similarity
- Multiple **query** vectors
- Separate **key** and **value**

Attention Layer

在注意力机制中，查询向量（Query）表示要关注或检索的目标，键向量（Key）表示要与查询向量进行匹配或比较的源，值向量（Value）表示要根据查询向量和键向量的匹配程度来加权求和的信息。

Inputs:

Query vectors: **Q** (Shape: N_Q x D_Q)

Input vectors: **X** (Shape: N_X x D_X)

Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)

Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)

Computation:

Key vectors: **K** = $\mathbf{XW_K}$ (Shape: N_X x D_Q)

Value Vectors: **V** = $\mathbf{XW_V}$ (Shape: N_X x D_V)

Similarities: E = $\mathbf{QK}^T / \sqrt{D_Q}$ (Shape: N_Q x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$

Attention weights: A = softmax(E, dim=1) (Shape: N_Q x N_X)

Output vectors: Y = A**V** (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j} \mathbf{V_j}$

X_1

X_2

X_3

Q_1 Q_2 Q_3 Q_4

Attention Layer

将键向量和值向量分开的好处，例如，在处理自然语言处理（NLP）任务时：

——查询向量来表示目标语言单词

——键向量来表示源语言单词

——值向量来表示源语言单词的嵌入向量

这样，模型就可以根据目标语言单词的查询向量，找到与之最相关的源语言单词（即键向量），然后根据这些源语言单词的嵌入向量（即值向量），计算出一个加权的输出，从而实现更准确和高效的翻译。

Inputs:

Query vectors: **Q** (Shape: N_Q x D_Q)

Input vectors: **X** (Shape: N_X x D_X)

Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)

Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)

Computation:

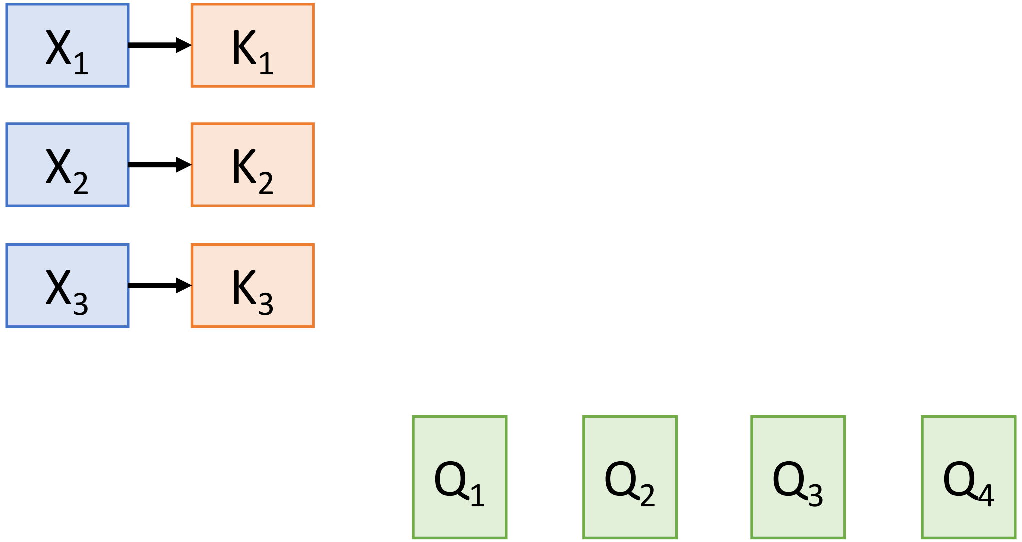
Key vectors: **K** = $\mathbf{XW_K}$ (Shape: N_X x D_Q)

Value Vectors: **V** = $\mathbf{XW_V}$ (Shape: N_X x D_V)

Similarities: E = $\mathbf{QK}^T / \sqrt{D_Q}$ (Shape: N_Q x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$

Attention weights: A = softmax(E, dim=1) (Shape: N_Q x N_X)

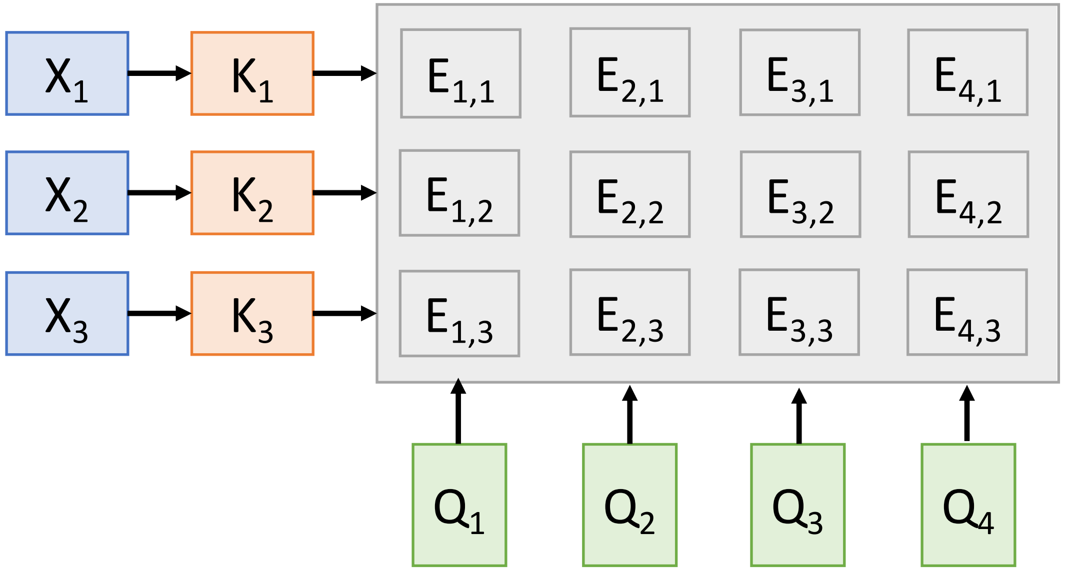
Output vectors: Y = A**V** (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$



Attention Layer

Inputs:
Query vectors: **Q** (Shape: N_Q x D_Q)
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)

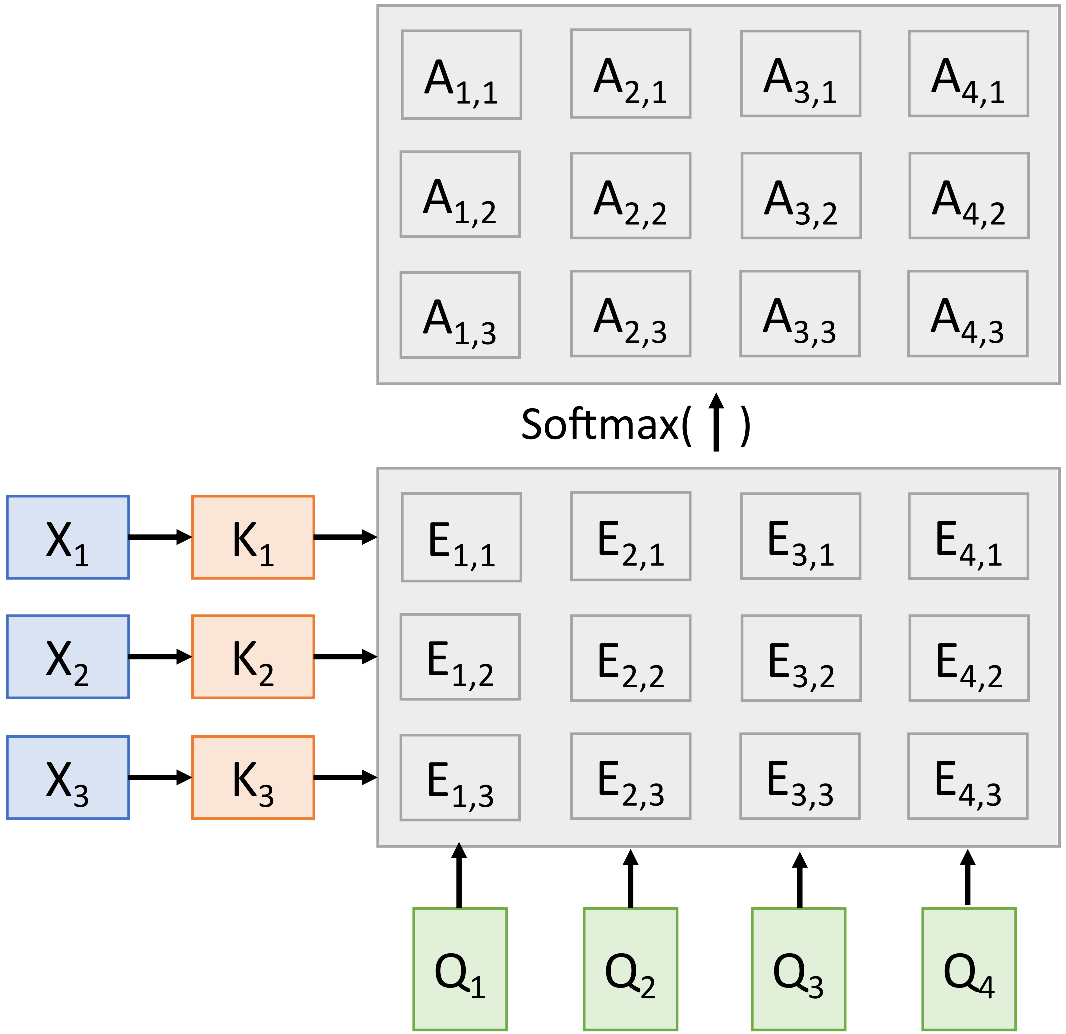
Computation:
Key vectors: **K** = $\mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: **V** = $\mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: E = $\mathbf{QK}^T / \sqrt{D_Q}$ (Shape: N_Q x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_Q x N_X)
Output vectors: Y = A**V** (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j} \mathbf{V_j}$



Attention Layer

Inputs:
Query vectors: **Q** (Shape: N_Q x D_Q)
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)

Computation:
Key vectors: **K** = $\mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: **V** = $\mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: E = $\mathbf{QK^T} / \sqrt{D_Q}$ (Shape: N_Q x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_Q x N_X)
Output vectors: Y = A**V** (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j} \mathbf{V_j}$

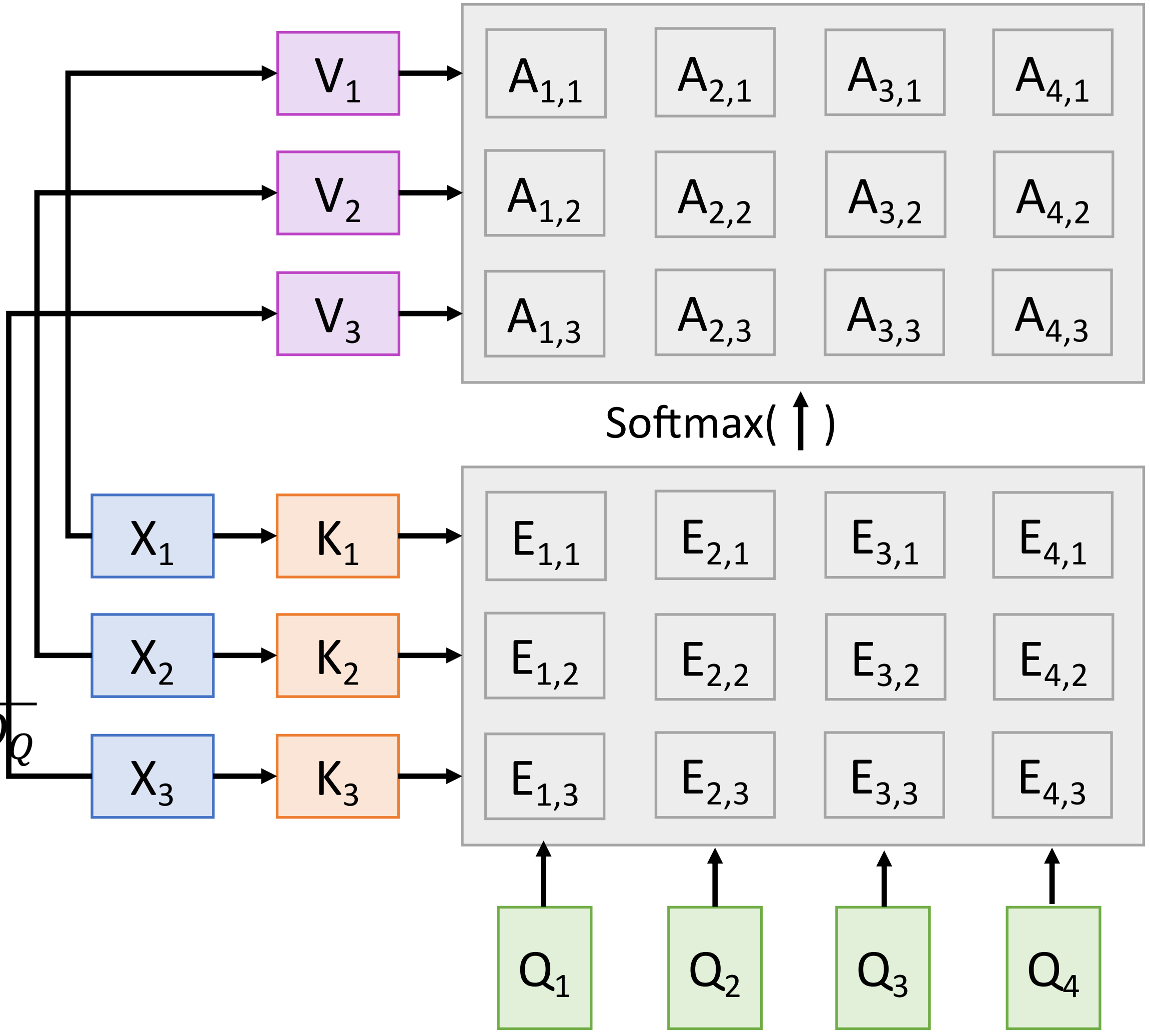


Attention Layer

具体来说，当dim=1时，是对某一维度的列进行softmax运算。例如，如果输入是一个二维张量（矩阵），那么dim=1就表示对每一行的元素进行softmax运算，使得每一行的元素之和为1。

Inputs:
Query vectors: **Q** (Shape: N_Q x D_Q)
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)

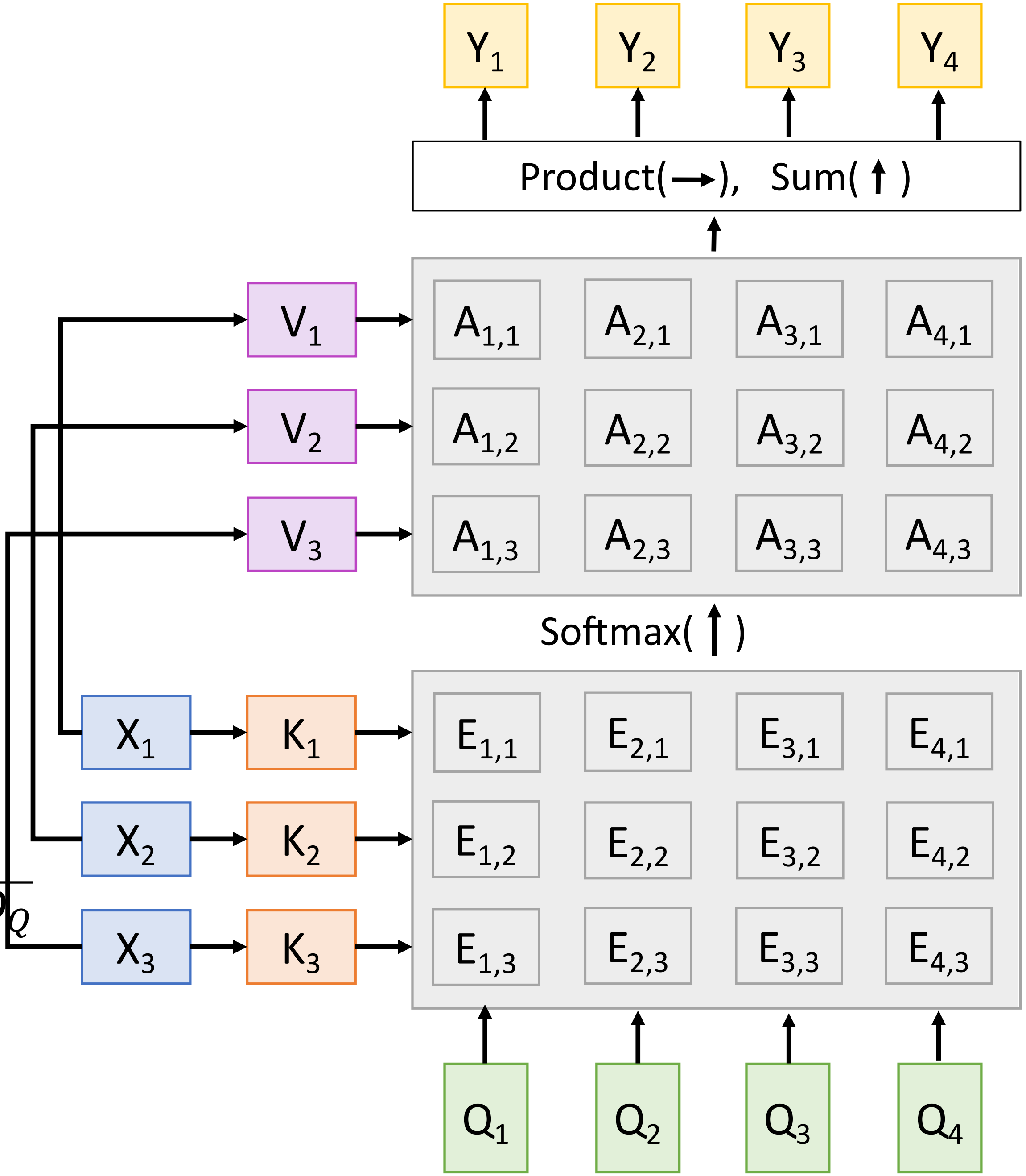
Computation:
Key vectors: **K** = **X**W_K (Shape: N_X x D_Q)
Value Vectors: **V** = **X**W_V (Shape: N_X x D_V)
Similarities: E = $QK^T / \sqrt{D_Q}$ (Shape: N_Q x N_X) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_Q x N_X)
Output vectors: Y = AV (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j} V_j$



Attention Layer

Inputs:
Query vectors: $\mathbf{Q}$ (Shape: N_Q x D_Q)
Input vectors: $\mathbf{X}$ (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)

Computation:
Key vectors: $\mathbf{K} = \mathbf{X}\mathbf{W_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{X}\mathbf{W_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{Q}\mathbf{K}^T / \sqrt{D_Q}$ (Shape: N_Q x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: N_Q x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$



Self-Attention Layer

One **query** per **input vector**

Inputs:
Query vectors: **Q** (Shape: N_Q x D_Q)
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)

注意力机制关注的是目标和源之间的关系，而自注意力机制关注的是源或目标内部元素之间的关系。

由于查询、键和值来自同一组输入，因此被称为 自注意力

Computation:
Key vectors: $\mathbf{K} = \mathbf{X}\mathbf{W_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{X}\mathbf{W_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{Q}\mathbf{K}^T / \sqrt{D_Q}$ (Shape: N_Q x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_Q x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$

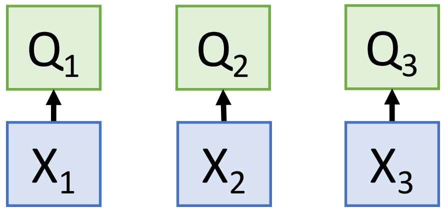
X_1 X_2 X_3

Self-Attention Layer

One **query** per **input vector**

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

Computation:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK^T} / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$

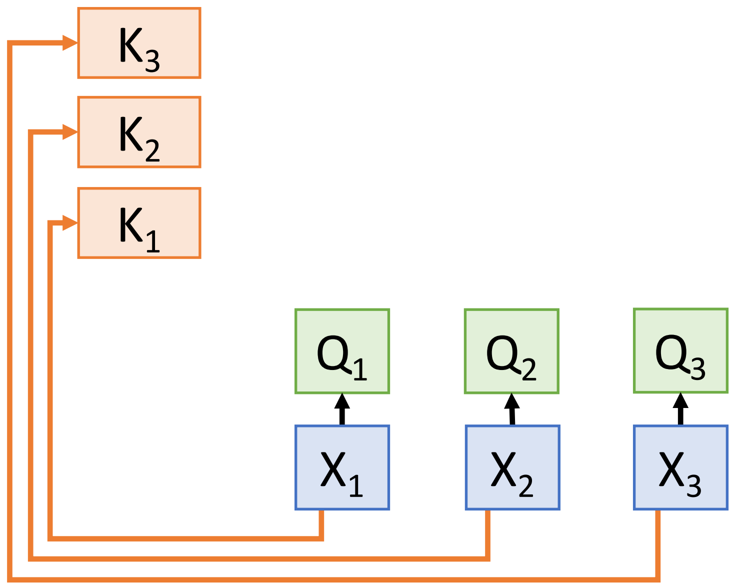


Self-Attention Layer

One **query** per **input vector**

<u>Inputs</u>:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

<u>Computation</u>:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: N_X x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$

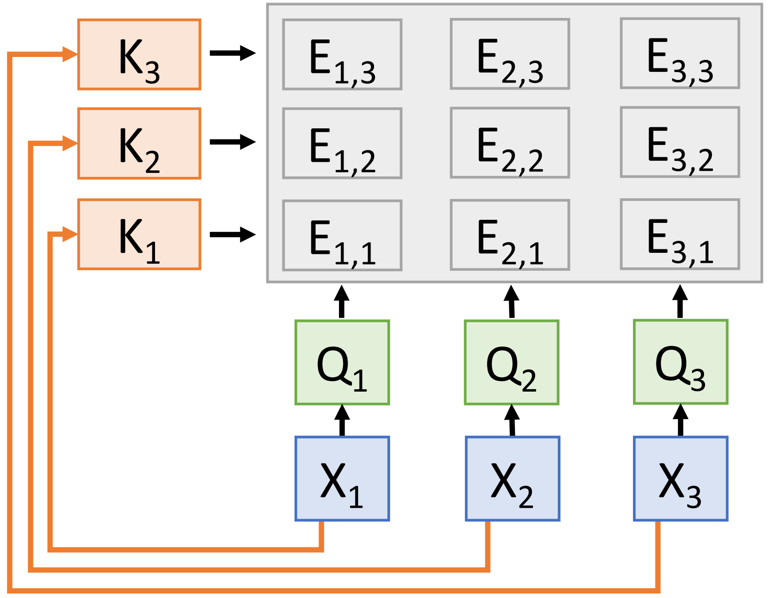


Self-Attention Layer

One **query** per **input vector**

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: **W_K** (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)
Query matrix: **W_Q** (Shape: D_X x D_Q)

Computation:
Query vectors: **Q** = **XW_Q**
Key vectors: **K** = **XW_K** (Shape: N_X x D_Q)
Value Vectors: **V** = **XW_V** (Shape: N_X x D_V)
Similarities: $E = QK^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: Y = A**V** (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$

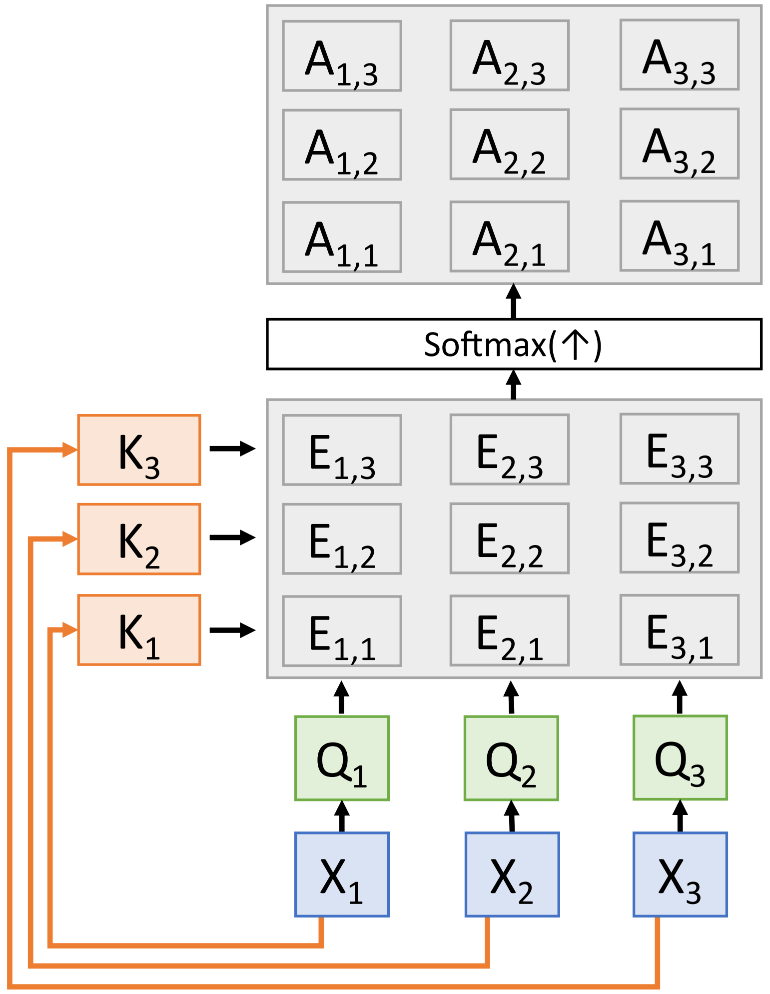


Self-Attention Layer

One **query** per **input vector**

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

Computation:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} \mathbf{V_j}$

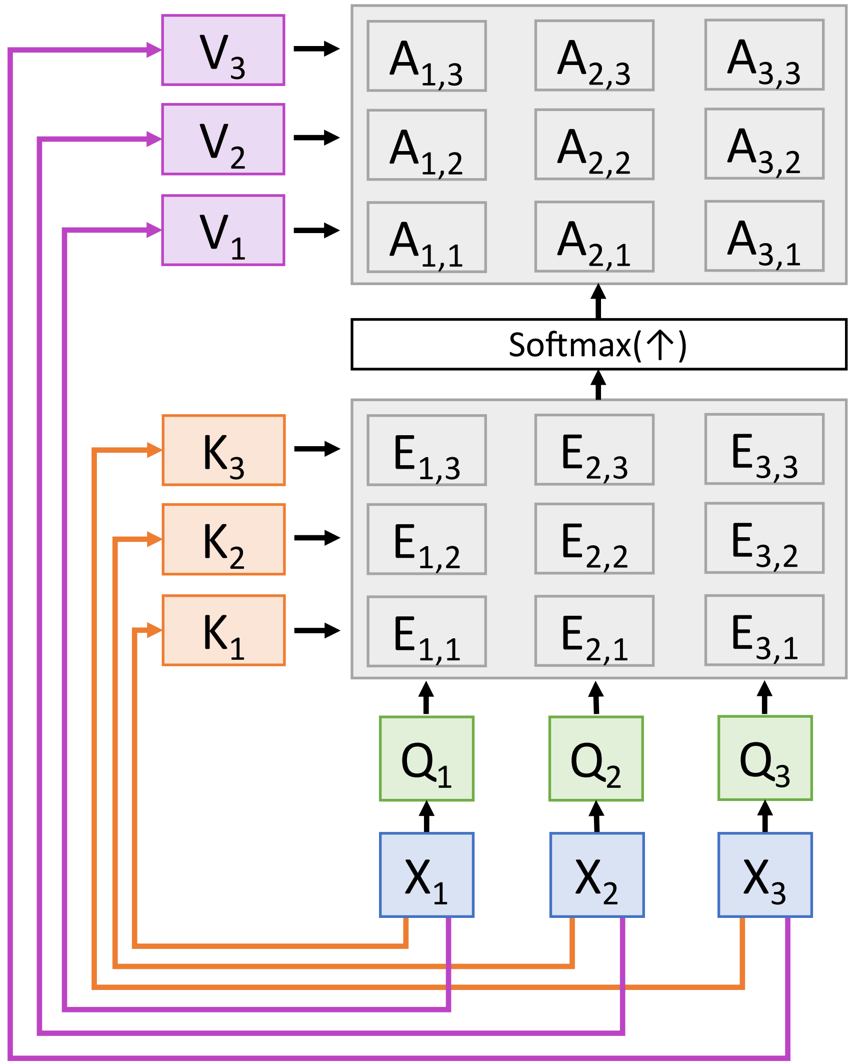


Self-Attention Layer

One **query** per **input vector**

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

Computation:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$

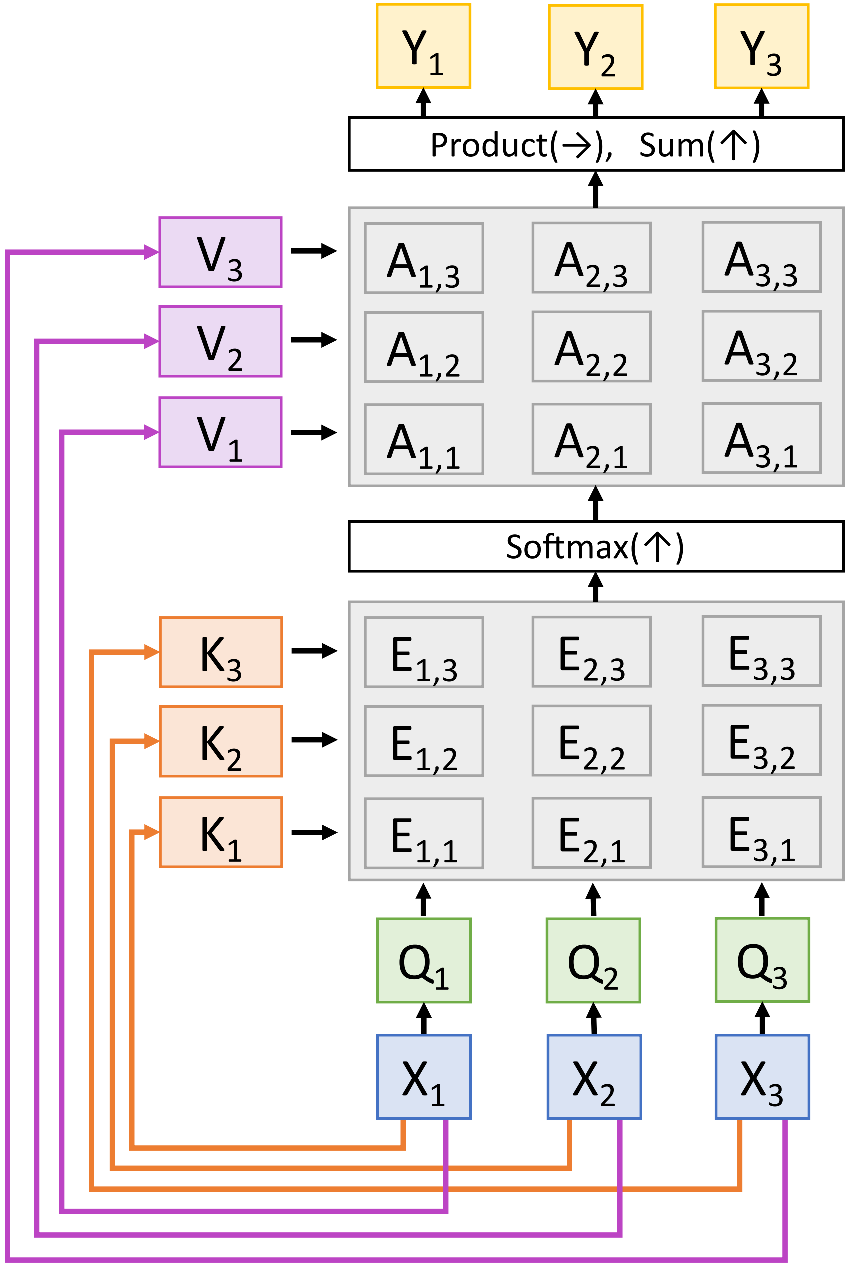


Self-Attention Layer

One **query** per **input vector**

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: **W_K** (Shape: D_X x D_Q)
Value matrix: **W_V** (Shape: D_X x D_V)
Query matrix: **W_Q** (Shape: D_X x D_Q)

Computation:
Query vectors: **Q** = **XW_Q**
Key vectors: **K** = **XW_K** (Shape: N_X x D_Q)
Value Vectors: **V** = **XW_V** (Shape: N_X x D_V)
Similarities: E = **QK^T** $/\sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: Y = A**V** (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$



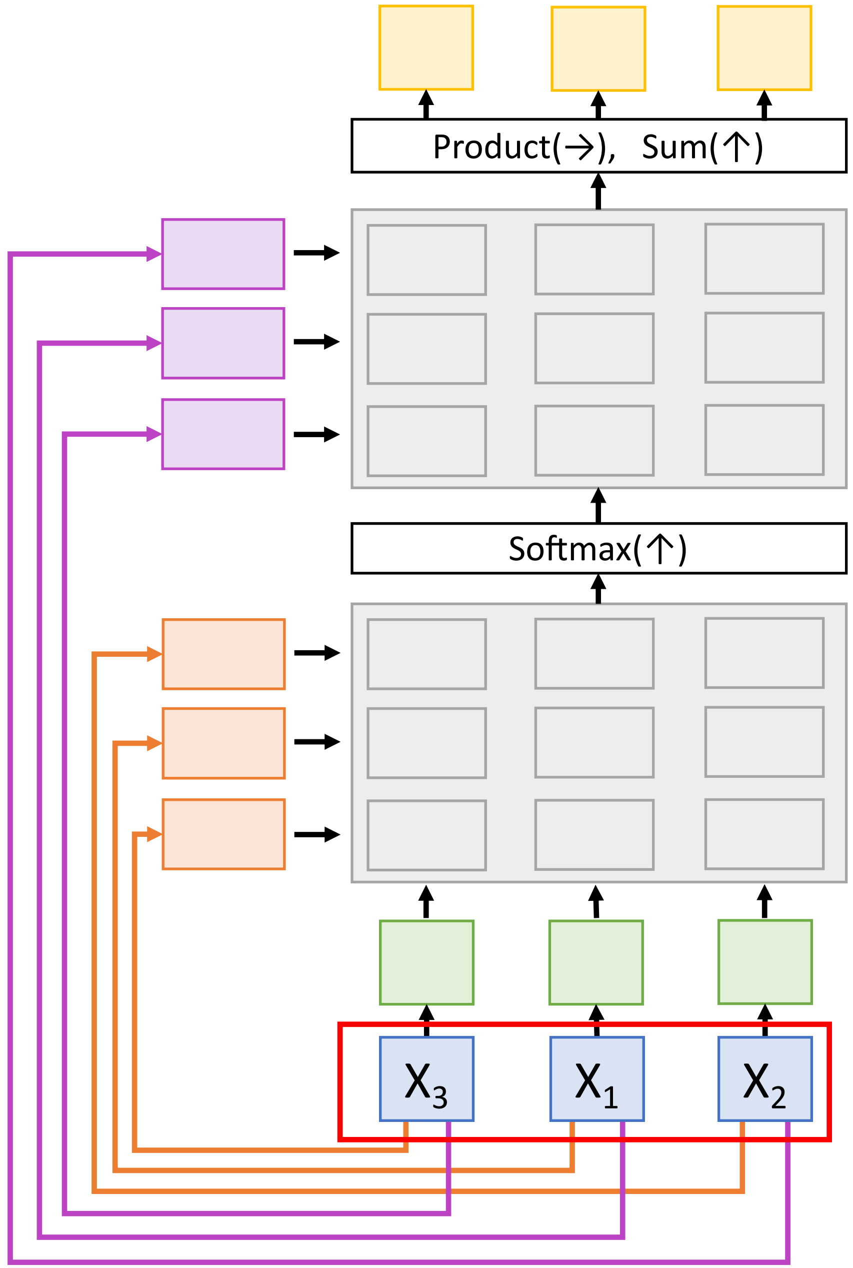
Self-Attention Layer

Consider **permuting** the input vectors:

考虑对输入向量进行排列

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

Computation:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$



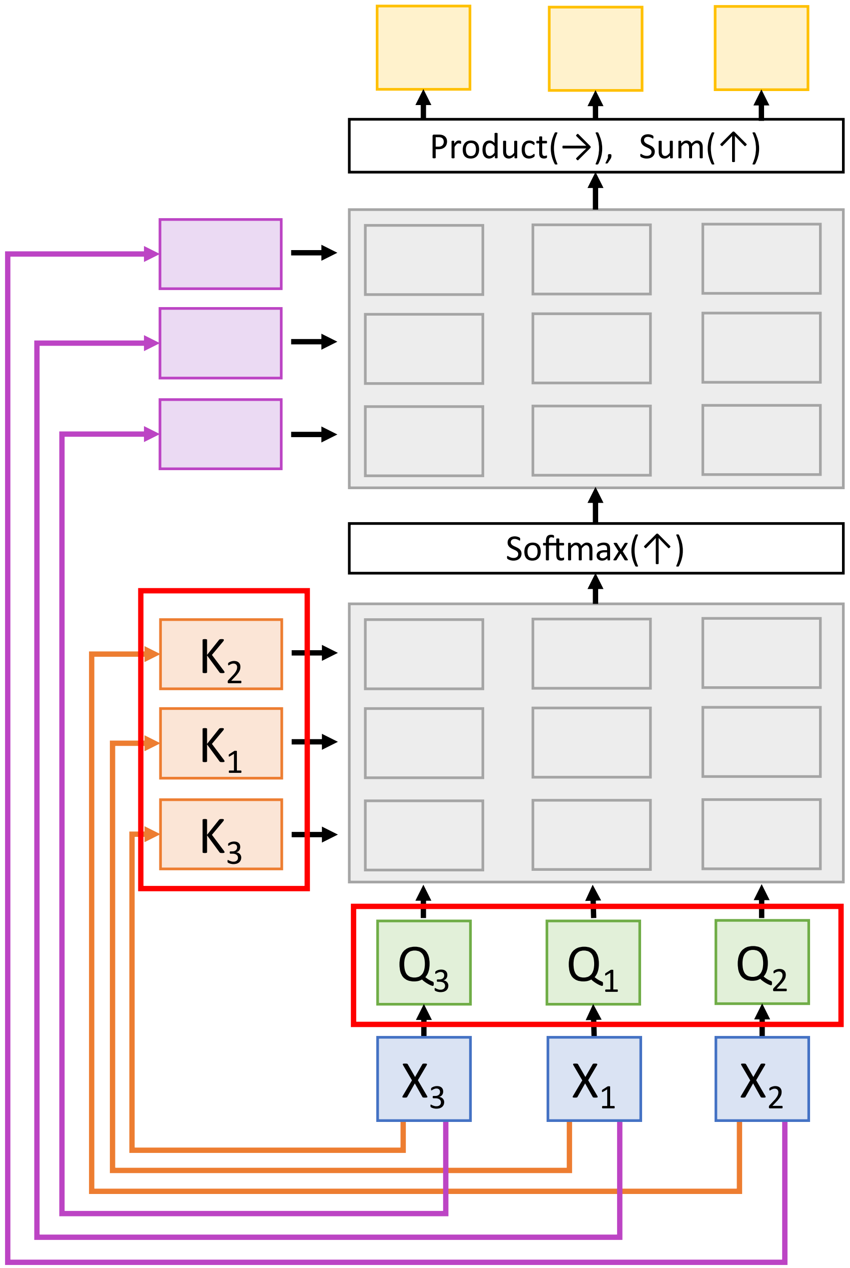
Self-Attention Layer

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

Computation:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} \mathbf{V_j}$

Consider **permuting** the input vectors:

Queries and Keys will be the same, but permuted



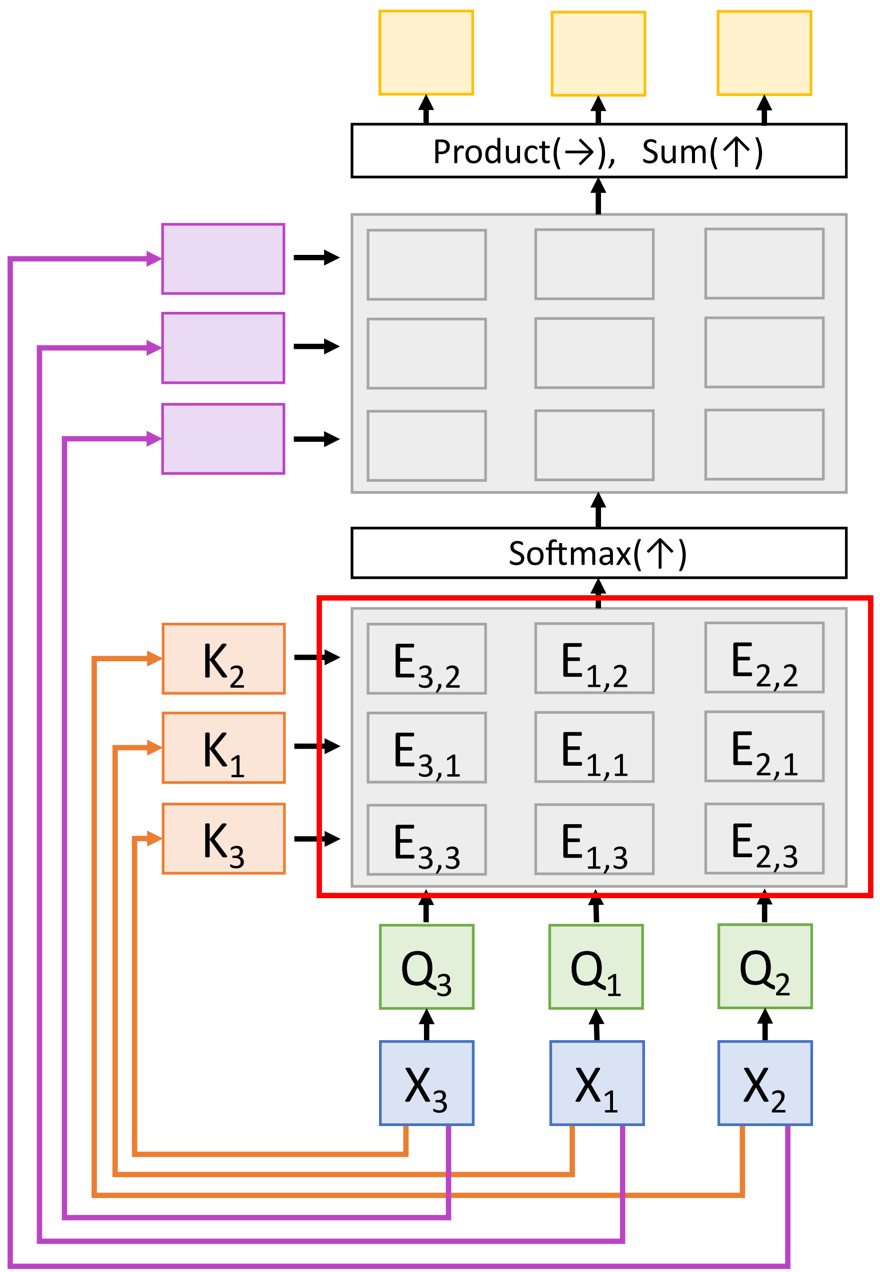
Self-Attention Layer

Consider **permuting** the input vectors:

Similarities will be the same, but permuted

<u>**Inputs**</u>:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: **W_K** (Shape: D_X x D_Q)
Value matrix: **W_V** (Shape: D_X x D_V)
Query matrix: **W_Q** (Shape: D_X x D_Q)

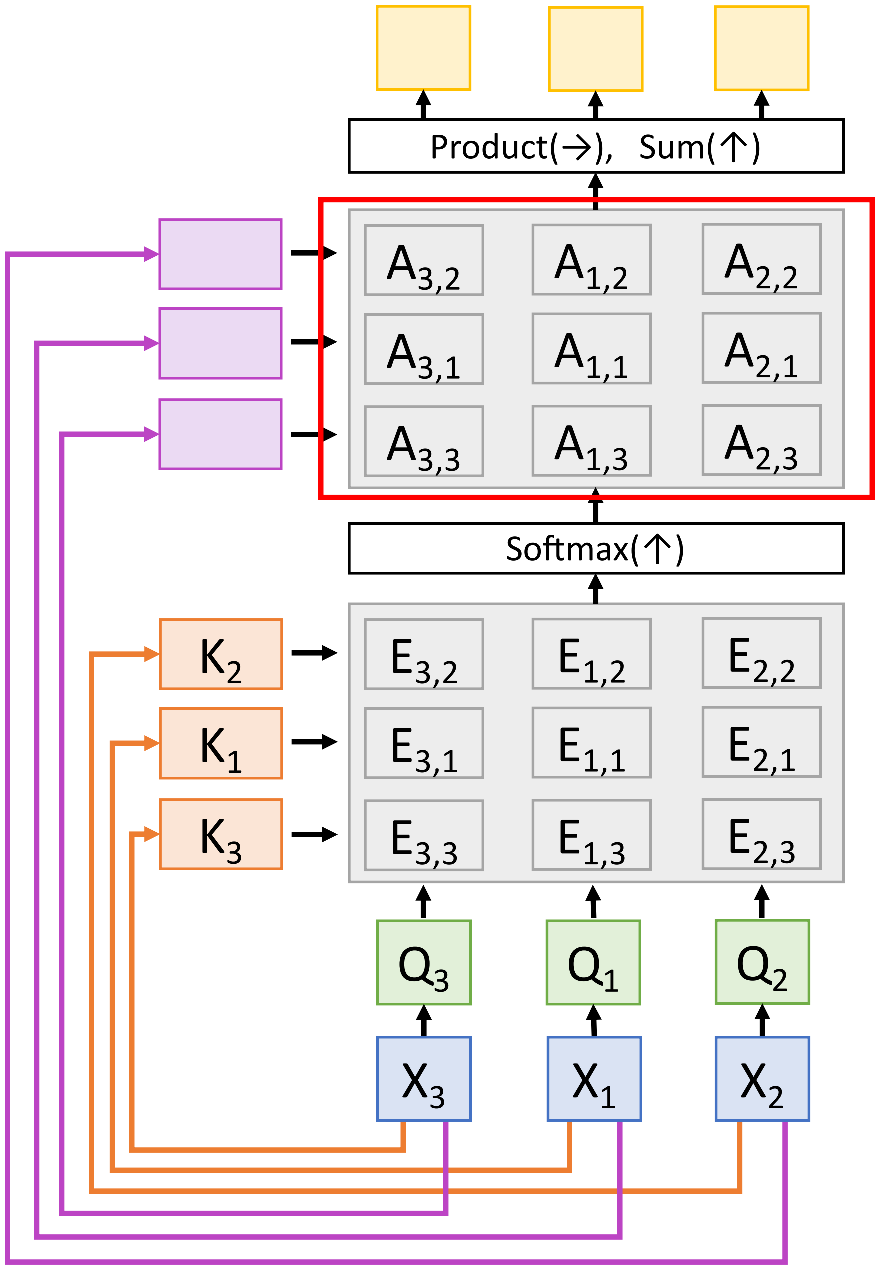
<u>**Computation**</u>:
Query vectors: **Q** = **XW_Q**
Key vectors: **K** = **XW_K** (Shape: N_X x D_Q)
Value Vectors: **V** = **XW_V** (Shape: N_X x D_V)
Similarities: $E = QK^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: Y = A**V** (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$



Self-Attention Layer

Consider **permuting** the input vectors:

Attention weights will be the same, but permuted



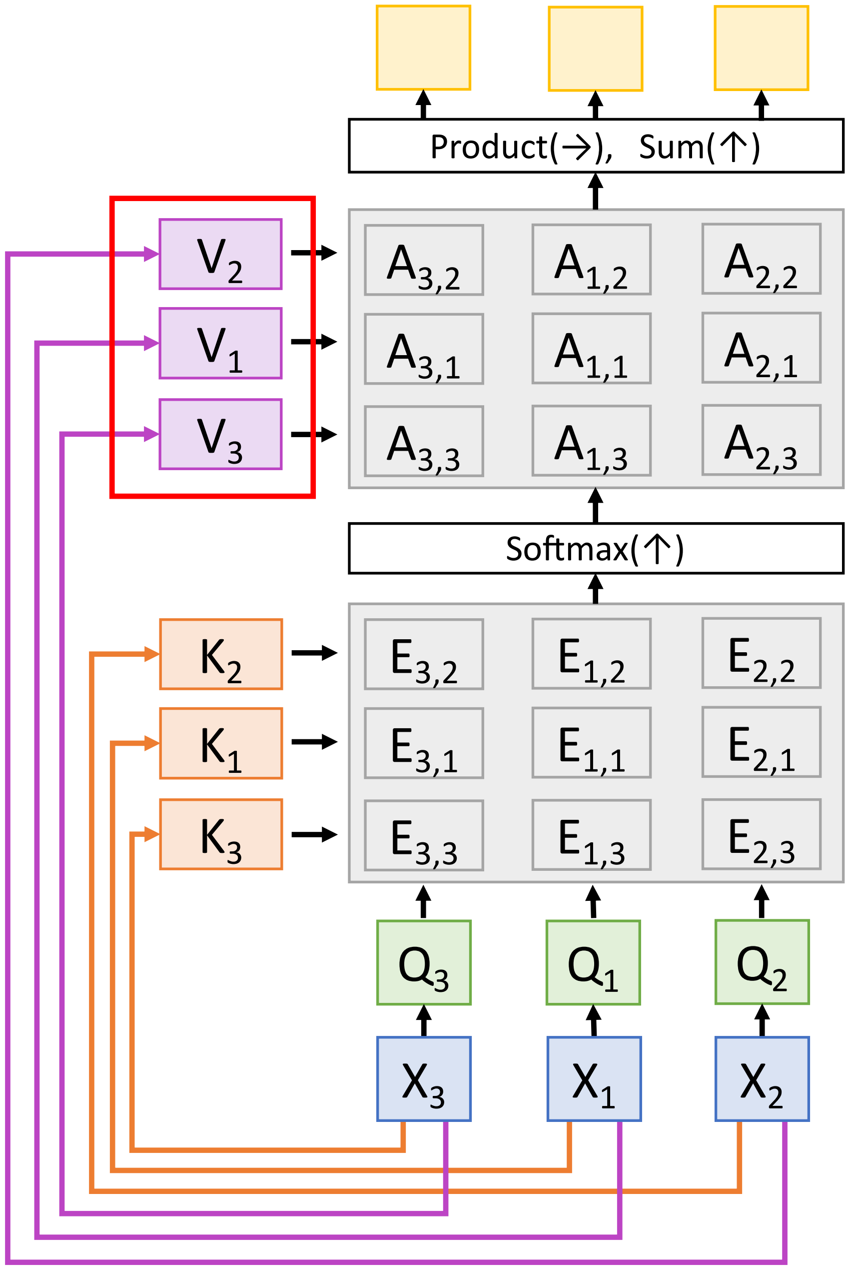
Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: **W_K** (Shape: D_X x D_Q)
Value matrix: **W_V** (Shape: D_X x D_V)
Query matrix: **W_Q** (Shape: D_X x D_Q)

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$

Self-Attention Layer

Consider **permuting** the input vectors:

Values will be the same, but permuted



<u>Inputs</u>:

Input vectors: **X** (Shape: N_X x D_X)

Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)

Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)

Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

<u>Computation</u>:

Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$

Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)

Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)

Similarities: $E = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$

Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)

Output vectors: $Y = A\mathbf{V}$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$

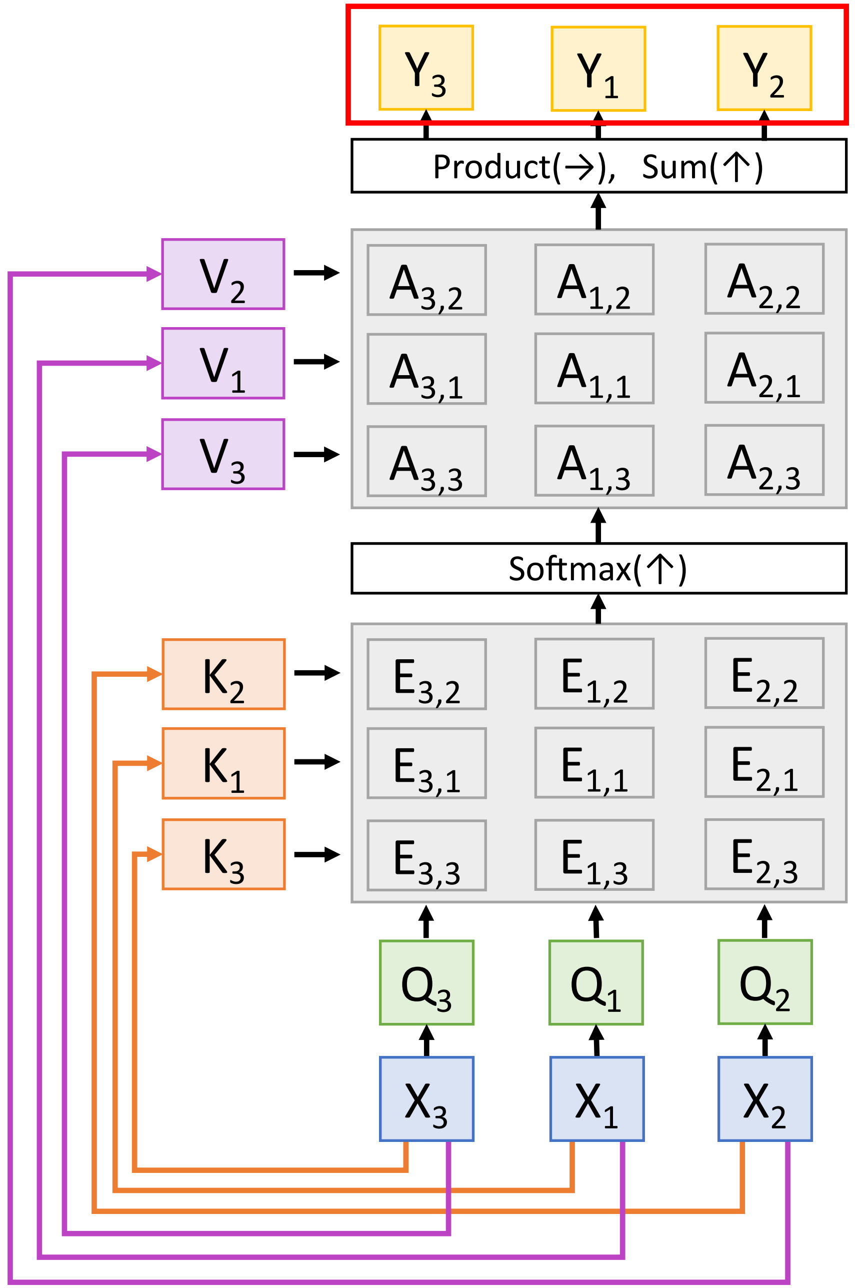
Self-Attention Layer

Consider **permuting** the input vectors:

Outputs will be the same, but permuted

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

Computation:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$



Self-Attention Layer

Inputs:
Input vectors: $\mathbf{X}$ (Shape: $N_X \times D_X$)
Key matrix: $\mathbf{W_K}$ (Shape: $D_X \times D_Q$)
Value matrix: $\mathbf{W_V}$ (Shape: $D_X \times D_V$)
Query matrix: $\mathbf{W_Q}$ (Shape: $D_X \times D_Q$)

置换同变性：改变输入数据顺序，输出顺序也会改变。

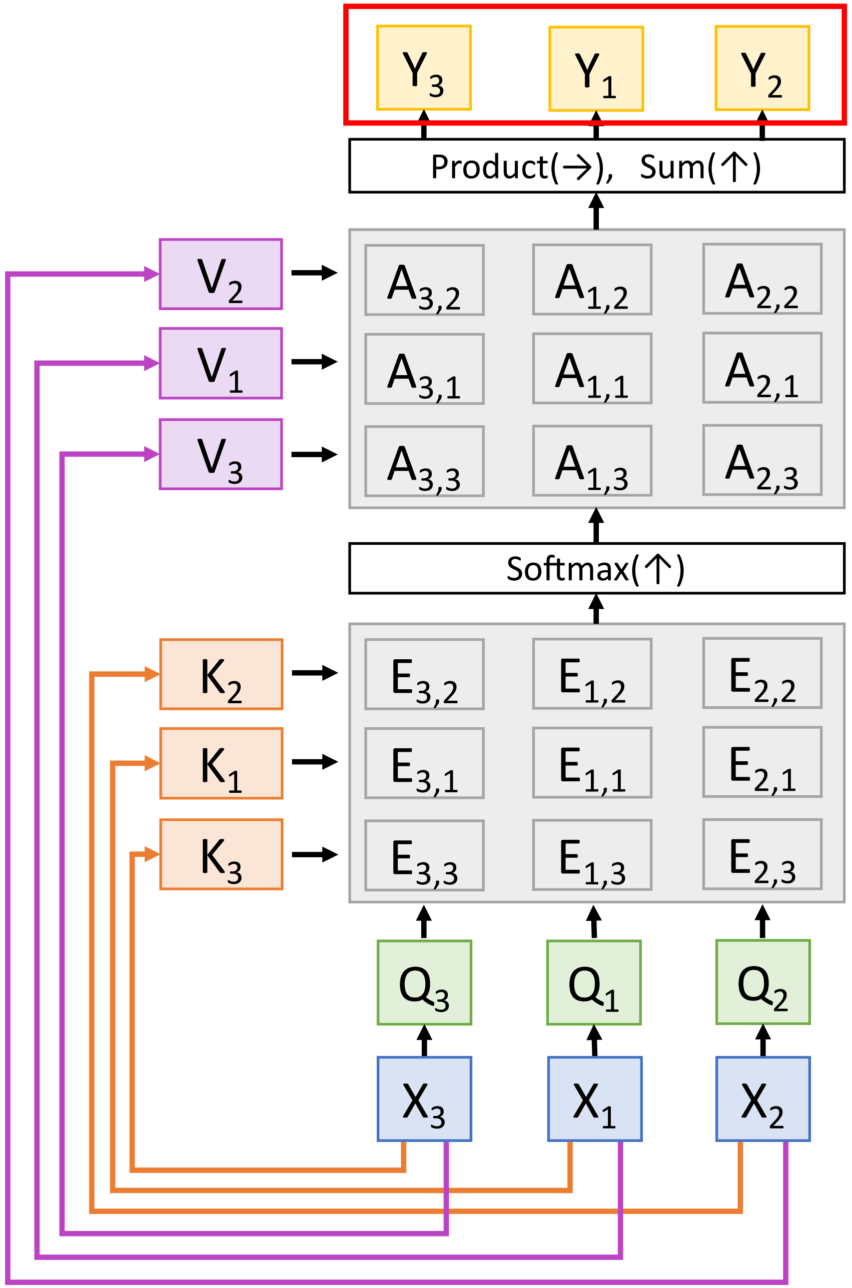
Computation:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: $N_X \times D_Q$)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: $N_X \times D_V$)
Similarities: $E = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_X \times N_X$)
Output vectors: $Y = A\mathbf{V}$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$

Consider **permuting** the input vectors:

Outputs will be the same, but permuted

Self-attention layer is **Permutation Equivariant** $f(s(x)) = s(f(x))$

Self-Attention layer works on **sets** of vectors

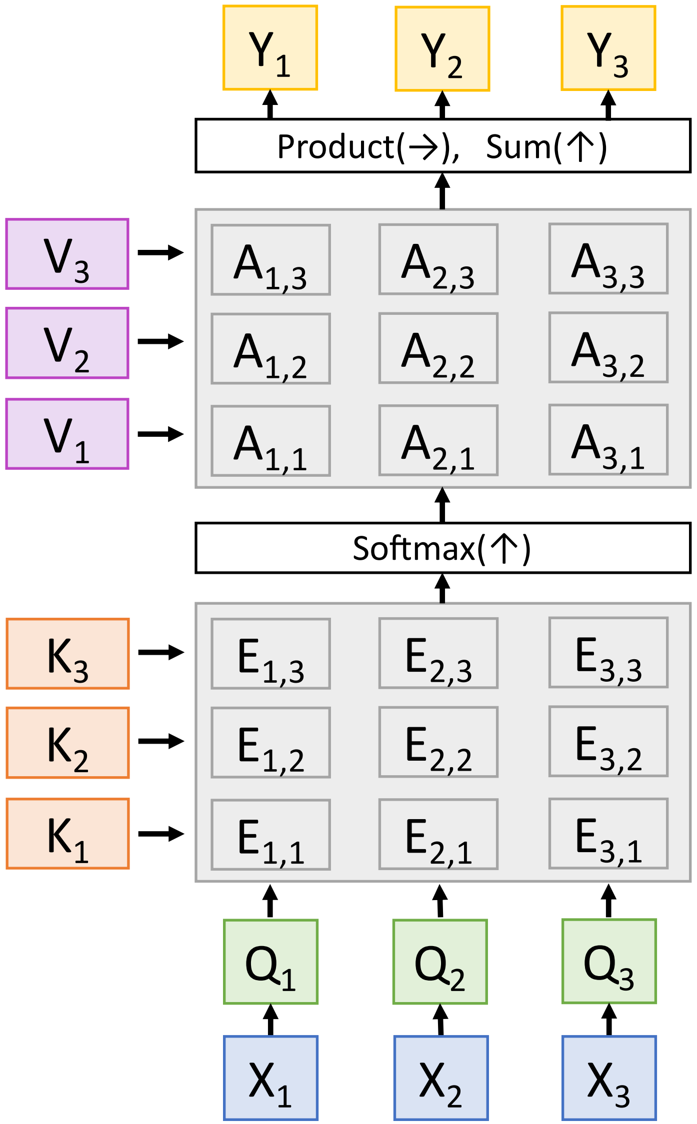


Self-Attention Layer

Self attention doesn't "know" the order of the vectors it is processing!

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

Computation:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$



Self-Attention Layer

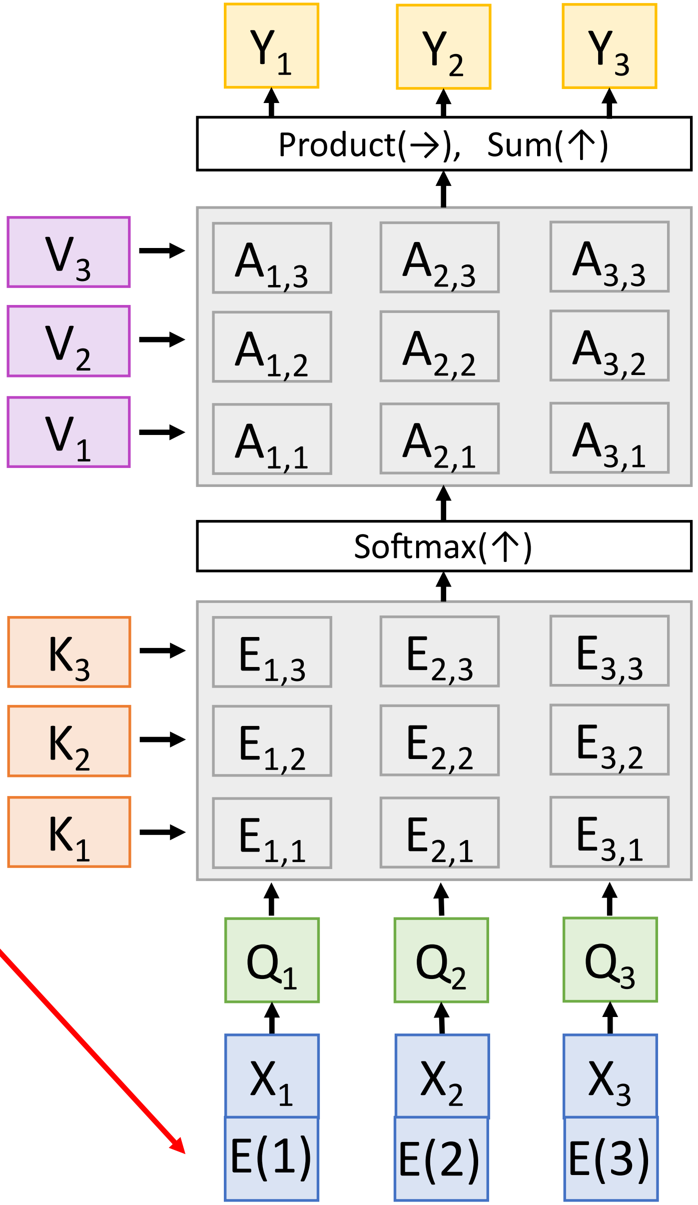
Inputs:
Input vectors: $\mathbf{X}$ (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

Computation:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: N_X x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$

Self attention doesn't "know" the order of the vectors it is processing!

In order to make processing position-aware, concatenate or add **positional encoding** to the input

E can be learned lookup table, or fixed function



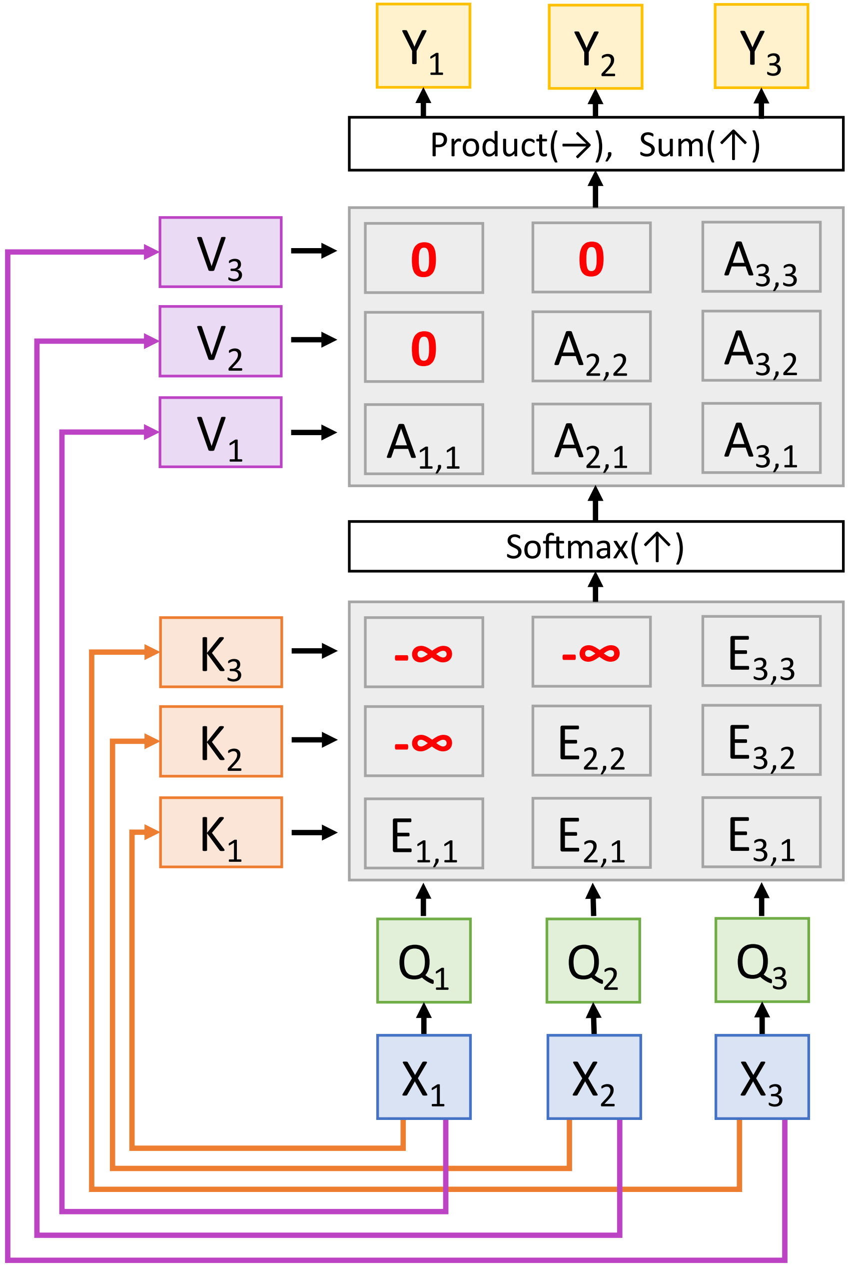
Masked Self-Attention Layer

Don't let vectors "look ahead" in the sequence

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: **W_K** (Shape: D_X x D_Q)
Value matrix: **W_V** (Shape: D_X x D_V)
Query matrix: **W_Q** (Shape: D_X x D_Q)

Computation:
Query vectors: **Q** = **XW_Q**
Key vectors: **K** = **XW_K** (Shape: N_X x D_Q)
Value Vectors: **V** = **XW_V** (Shape: N_X x D_V)
Similarities: E = $QK^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: Y = AV (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$

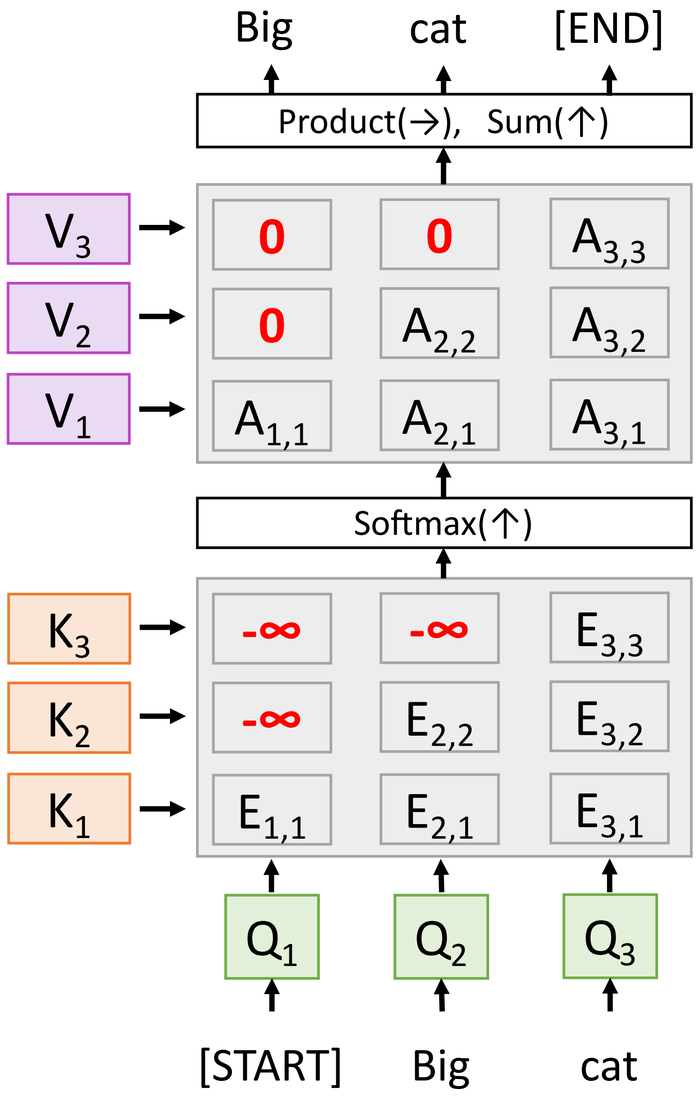
某些文本序列被填充了没有意义的特殊词元。为了仅将有意义的词元作为值来获取注意力汇聚，可以指定一个有效序列长度，以便在计算softmax时过滤掉超出指定范围的位置



Masked Self-Attention Layer

Don't let vectors "look ahead" in the sequence

Used for language modeling (predict next word)



Inputs:

Input vectors: **X** (Shape: N_X x D_X)

Key matrix: W_K (Shape: D_X x D_Q)

Value matrix: W_V (Shape: D_X x D_V)

Query matrix: W_Q (Shape: D_X x D_Q)

Computation:

Query vectors: $Q = XW_Q$

Key vectors: $K = XW_K$ (Shape: N_X x D_Q)

Value Vectors: $V = XW_V$ (Shape: N_X x D_V)

Similarities: $E = QK^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$

Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)

Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$

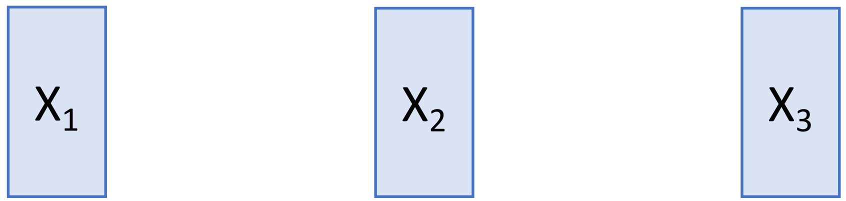
Multihead Self-Attention

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

Use H independent "Attention Heads" in parallel

Computation:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$

多头注意力机制（Multi-Head Attention）是一种在Transformer模型中被广泛采用的注意力机制扩展形式。其基本思想是通过并行地运行多个独立的注意力机制来获取输入序列的不同子空间的注意力分布，从而更全面地捕获序列中潜在的多种语义关联。



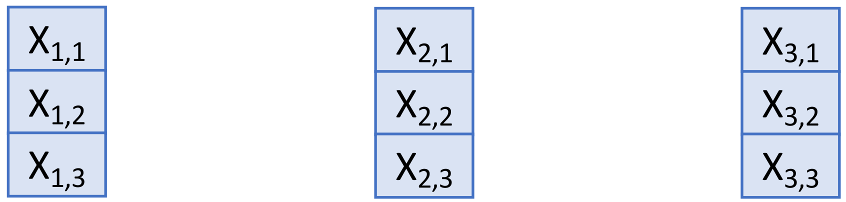
Multihead Self-Attention

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

Use H independent "Attention Heads" in parallel

Computation:
Query vectors: **Q** = $\mathbf{XW_Q}$
Key vectors: **K** = $\mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: **V** = $\mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: E = $\mathbf{QK}^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: Y = A**V** (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$

Split

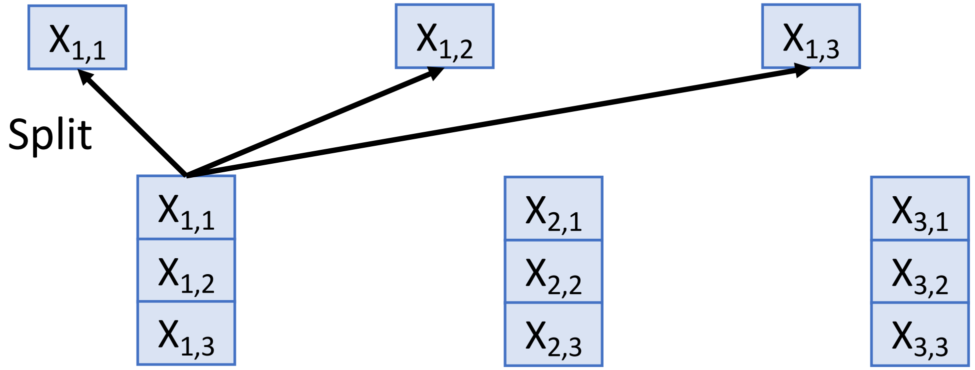


Multihead Self-Attention

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)
Query matrix: W_Q (Shape: D_X x D_Q)

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$

Use H independent "Attention Heads" in parallel

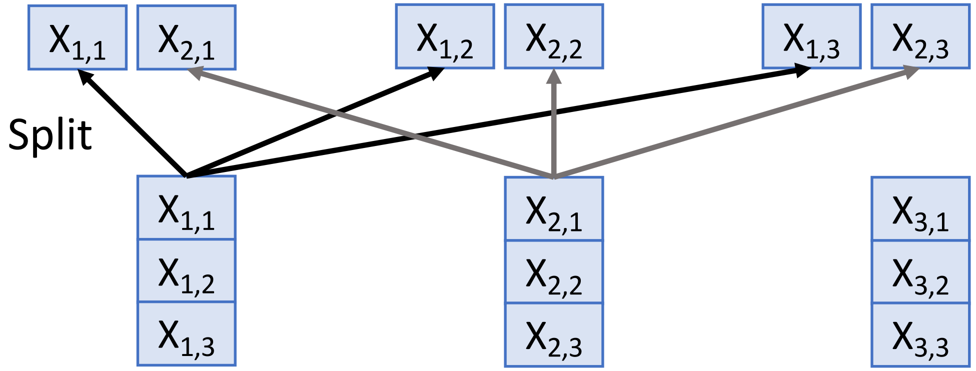


Multihead Self-Attention

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

Use H independent "Attention Heads" in parallel

Computation:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$



Multihead Self-Attention

Inputs:

Input vectors: **X** (Shape: N_X x D_X)

Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)

Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)

Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

Computation:

Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$

Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)

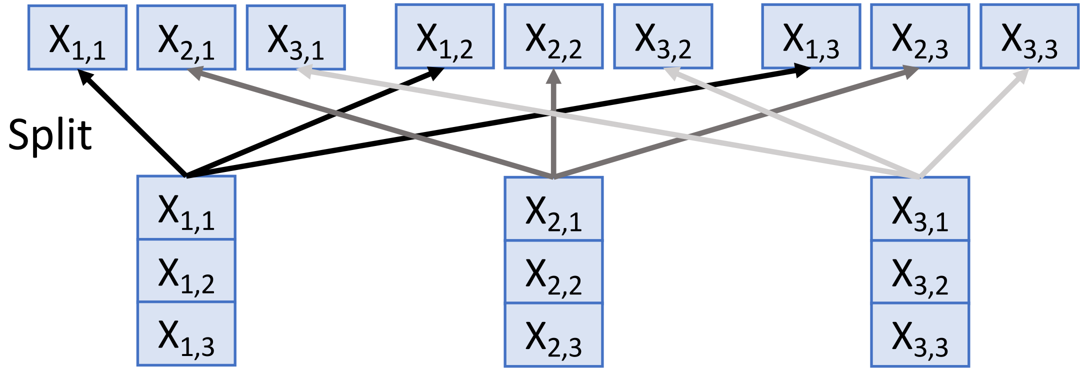
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)

Similarities: $E = \mathbf{QK^T} / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$

Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)

Output vectors: $Y = A\mathbf{V}$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$

Use H independent "Attention Heads" in parallel



Multihead Self-Attention

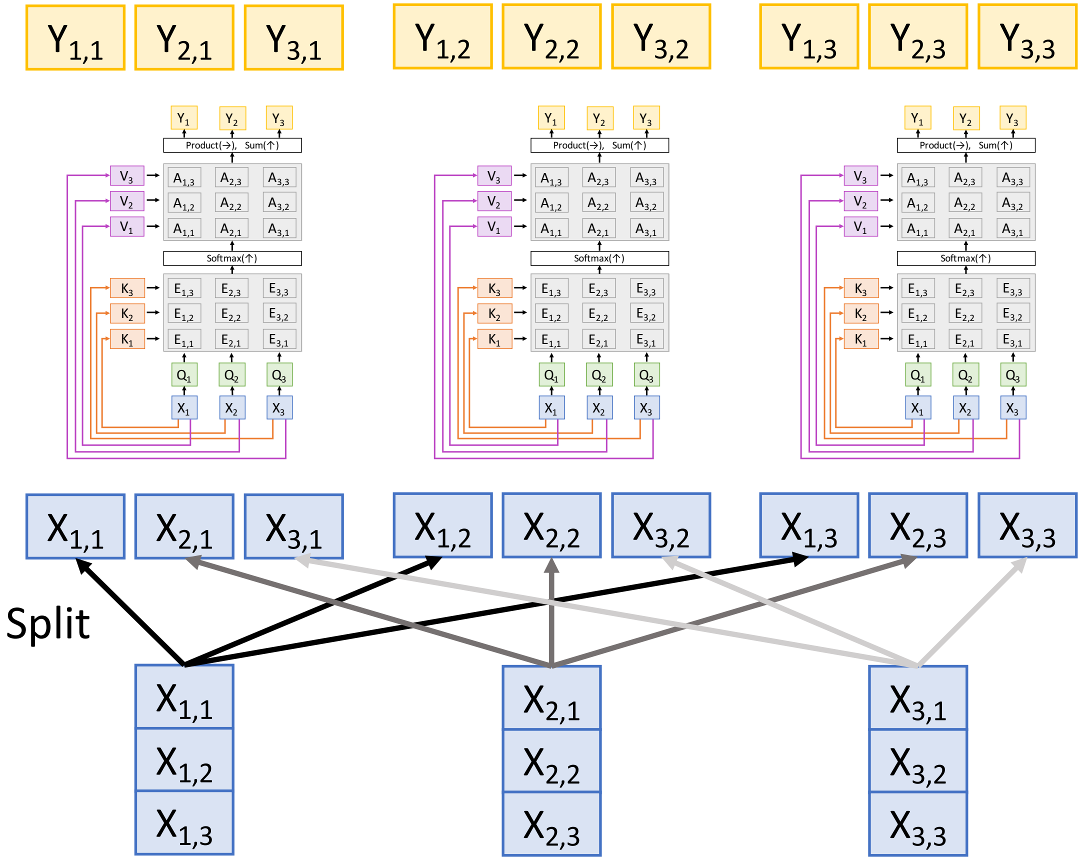
将输入的向量映射到多个不同的子空间，多头注意力并行学习，发现不同语义对应的信息，从而提取更多的特征

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

Computation:
Query vectors: **Q** = $\mathbf{XW_Q}$
Key vectors: **K** = $\mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: **V** = $\mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: E = $\mathbf{QK^T} / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: Y = A**V** (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} \mathbf{V_j}$

Use H independent "Attention Heads" in parallel

Run self-attention in parallel on each set of input vectors (different weights per head)

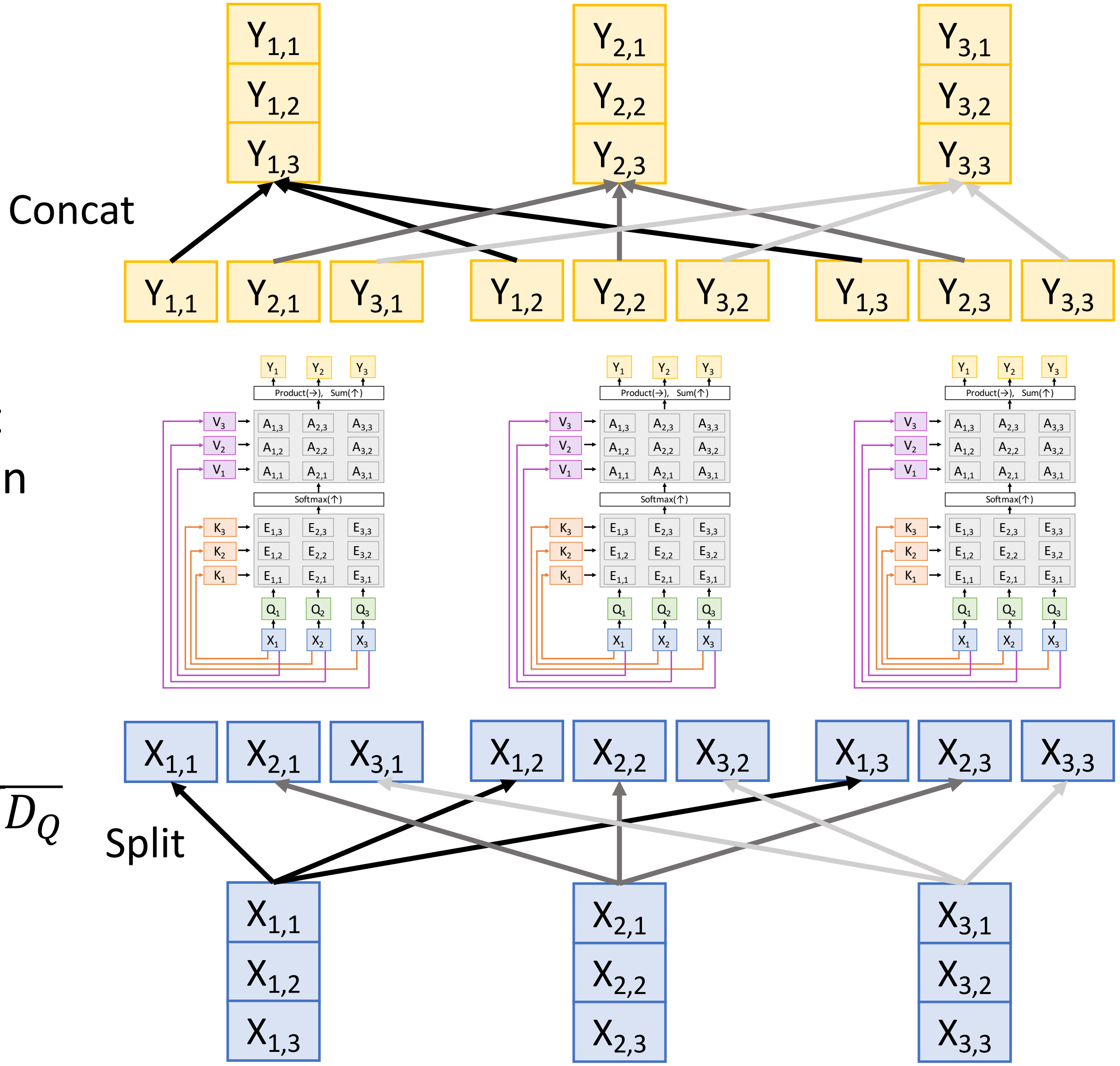


Multihead Self-Attention

Inputs:
Input vectors: **X** (Shape: $N_X \times D_X$)
Key matrix: $\mathbf{W_K}$ (Shape: $D_X \times D_Q$)
Value matrix: $\mathbf{W_V}$ (Shape: $D_X \times D_V$)
Query matrix: $\mathbf{W_Q}$ (Shape: $D_X \times D_Q$)

Computation:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: $N_X \times D_Q$)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: $N_X \times D_V$)
Similarities: $E = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_X \times N_X$)
Output vectors: $Y = A\mathbf{V}$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V_j}$

Use H independent "Attention Heads" in parallel

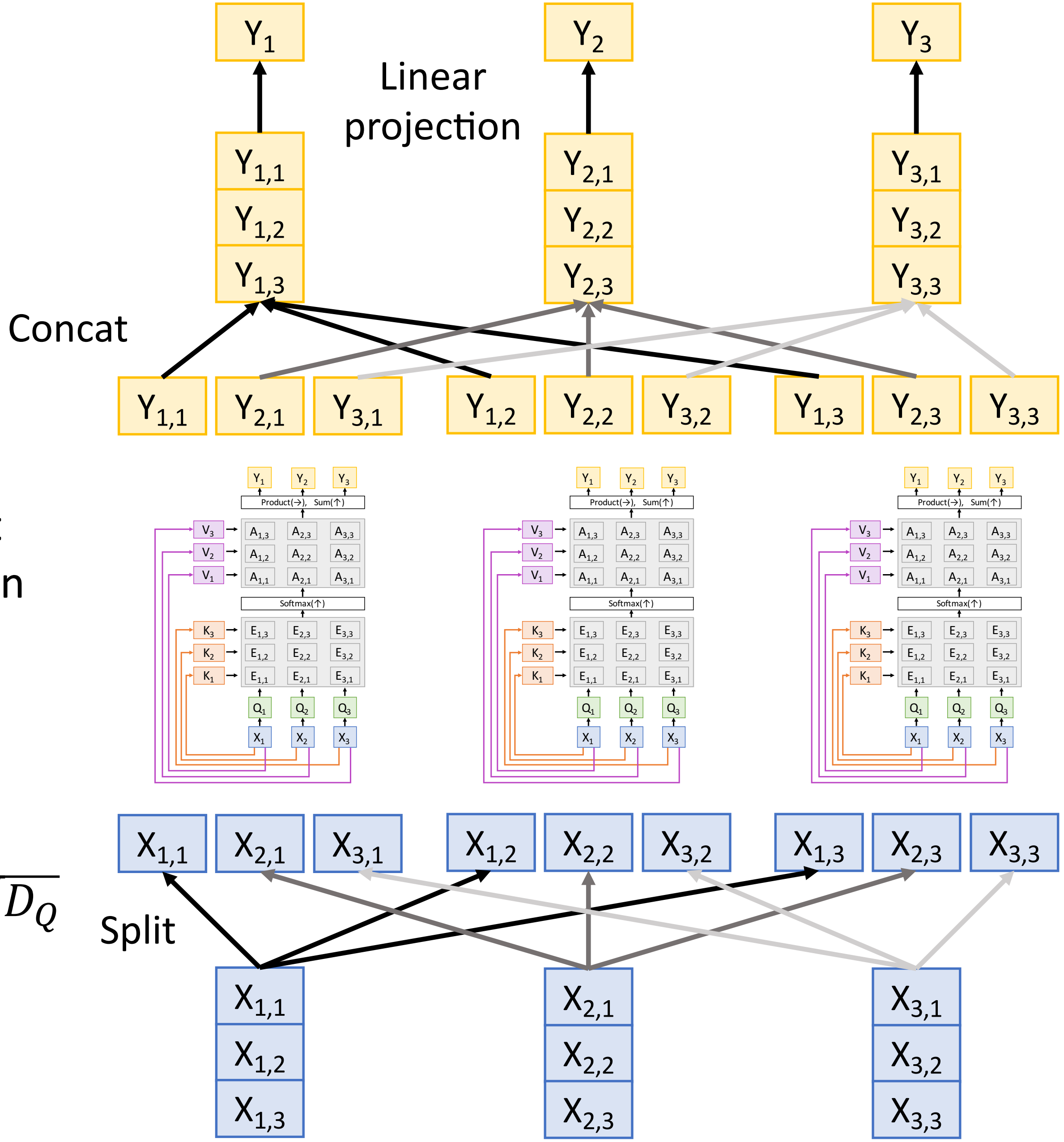


Multihead Self-Attention

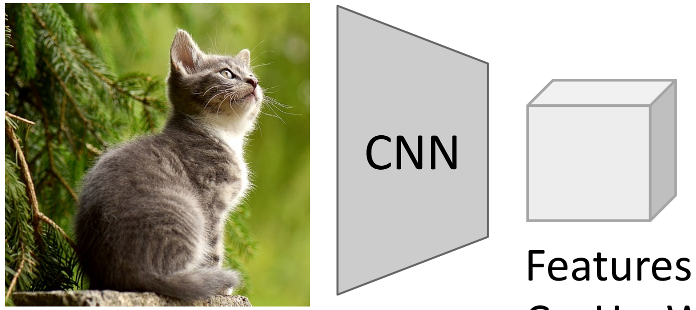
Inputs:
Input vectors: X (Shape: $N_X \times D_X$)
Key matrix: W_K (Shape: $D_X \times D_Q$)
Value matrix: W_V (Shape: $D_X \times D_V$)
Query matrix: W_Q (Shape: $D_X \times D_Q$)

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: $N_X \times D_Q$)
Value Vectors: $V = XW_V$ (Shape: $N_X \times D_V$)
Similarities: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_X \times N_X$)
Output vectors: $Y = AV$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} V_j$

Use H independent "Attention Heads" in parallel

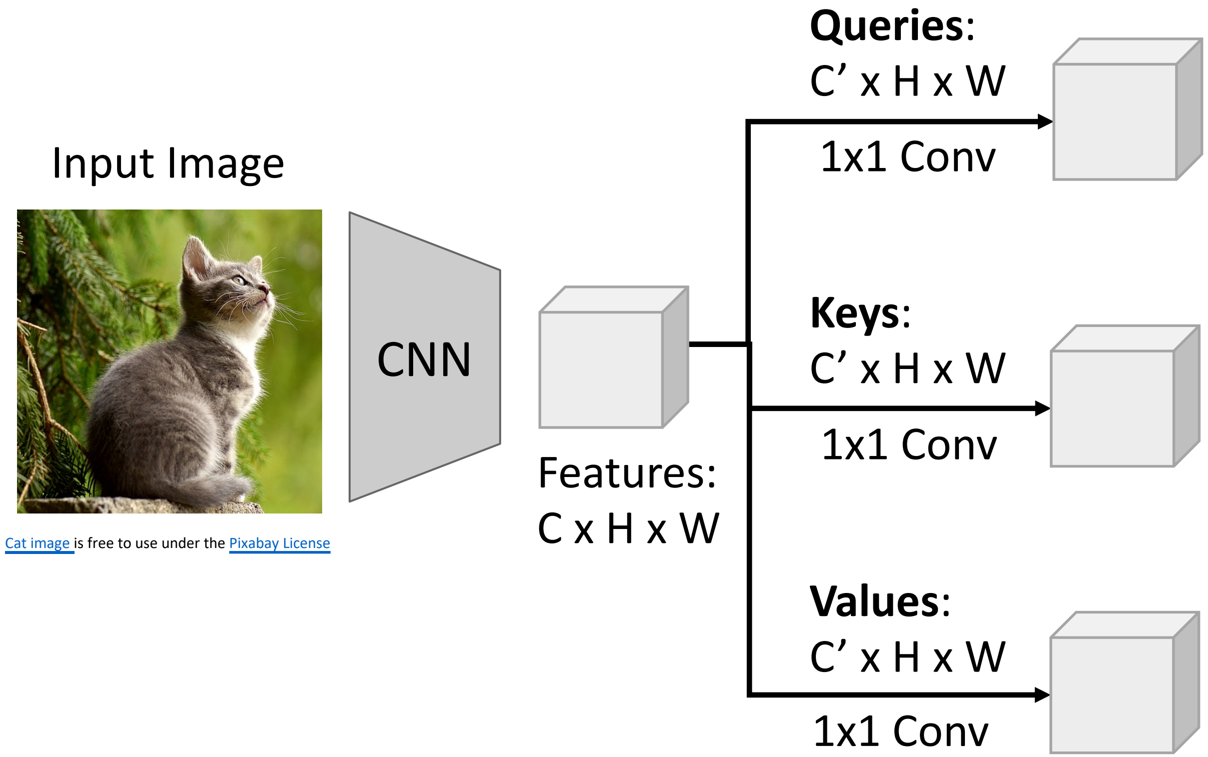


Example: CNN with Self-Attention



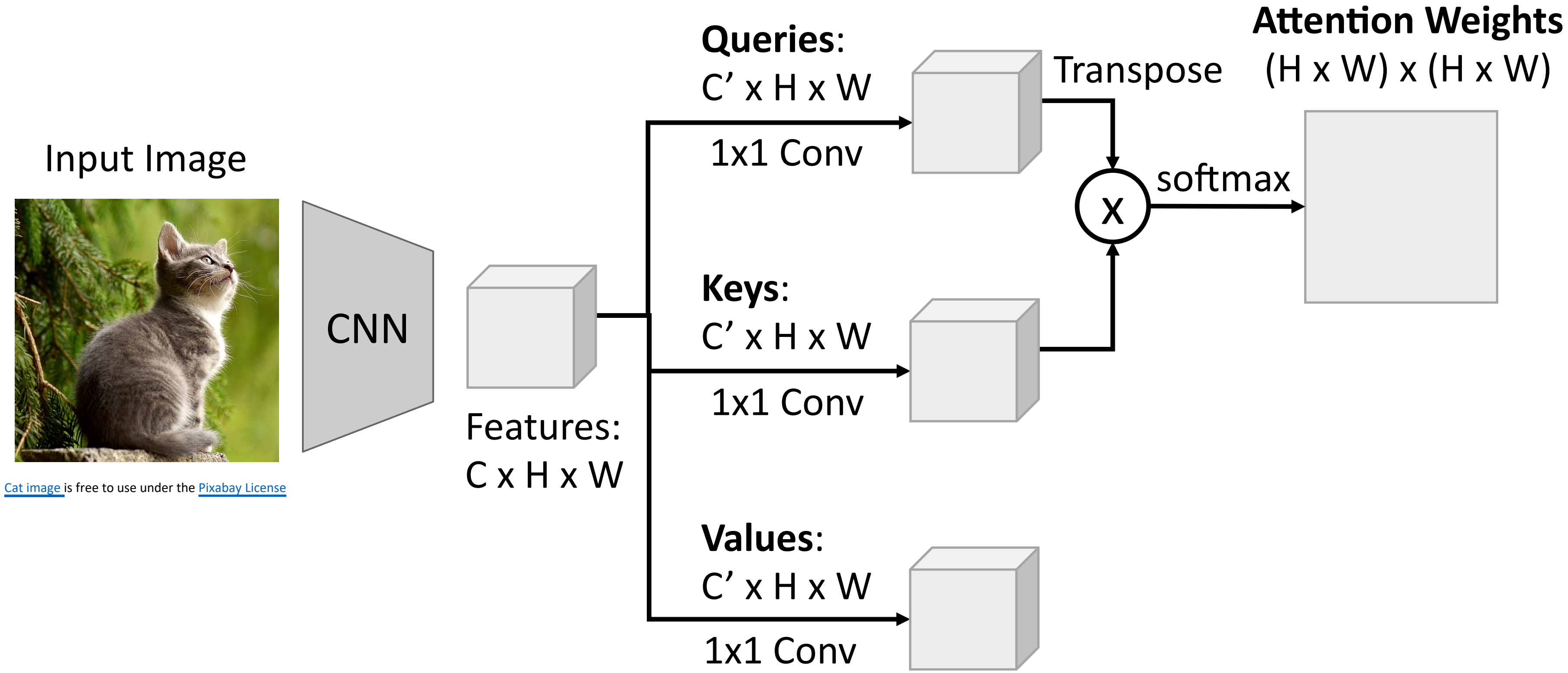
Zhang et al, “Self-Attention Generative Adversarial Networks”, ICML 2018

Example: CNN with Self-Attention



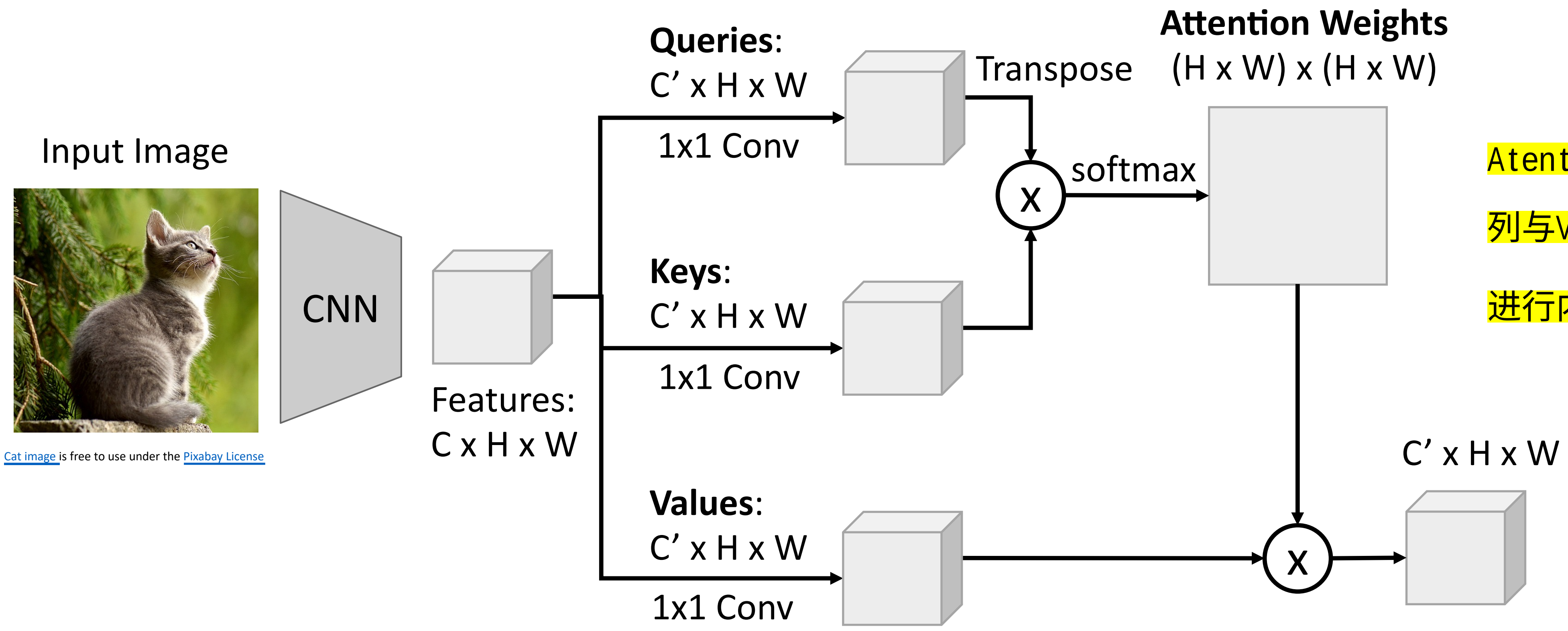
Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2018

Example: CNN with Self-Attention



Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2018

Example: CNN with Self-Attention

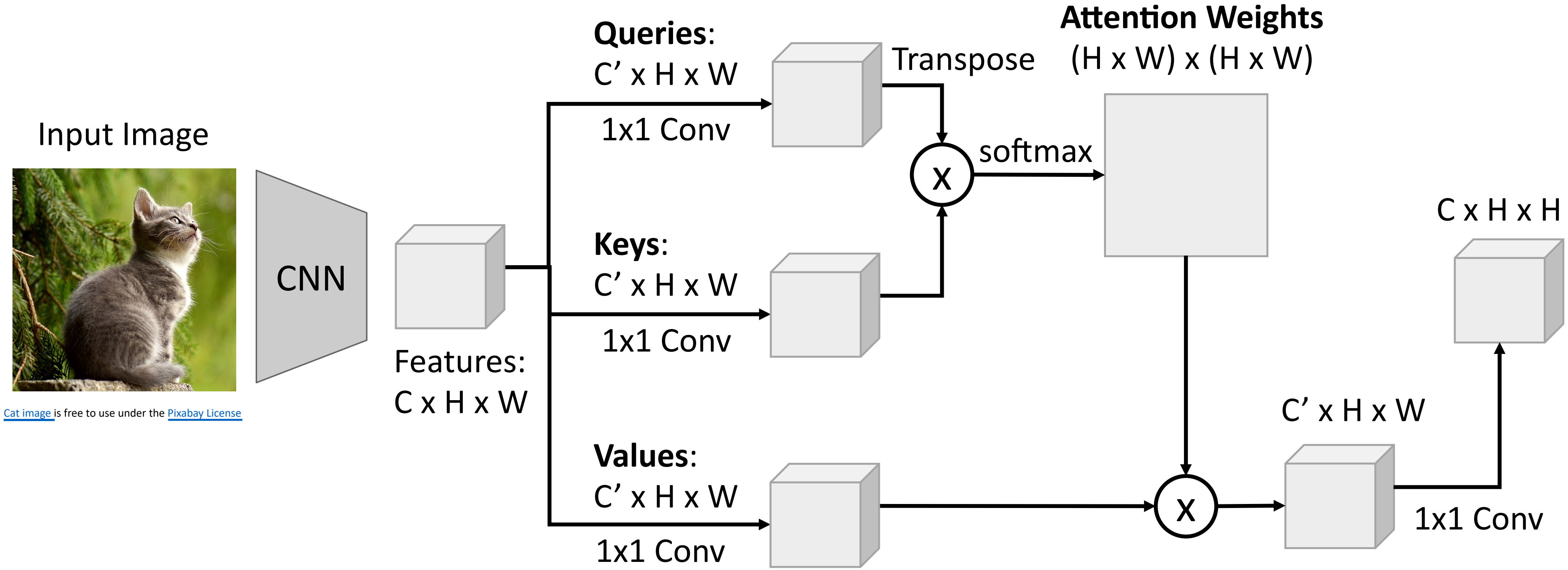


Cat image is free to use under the Pixabay License

Atention Weights中的每一列与Values中的每个channel进行内积，共得到HxW个值

Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2018

Example: CNN with Self-Attention

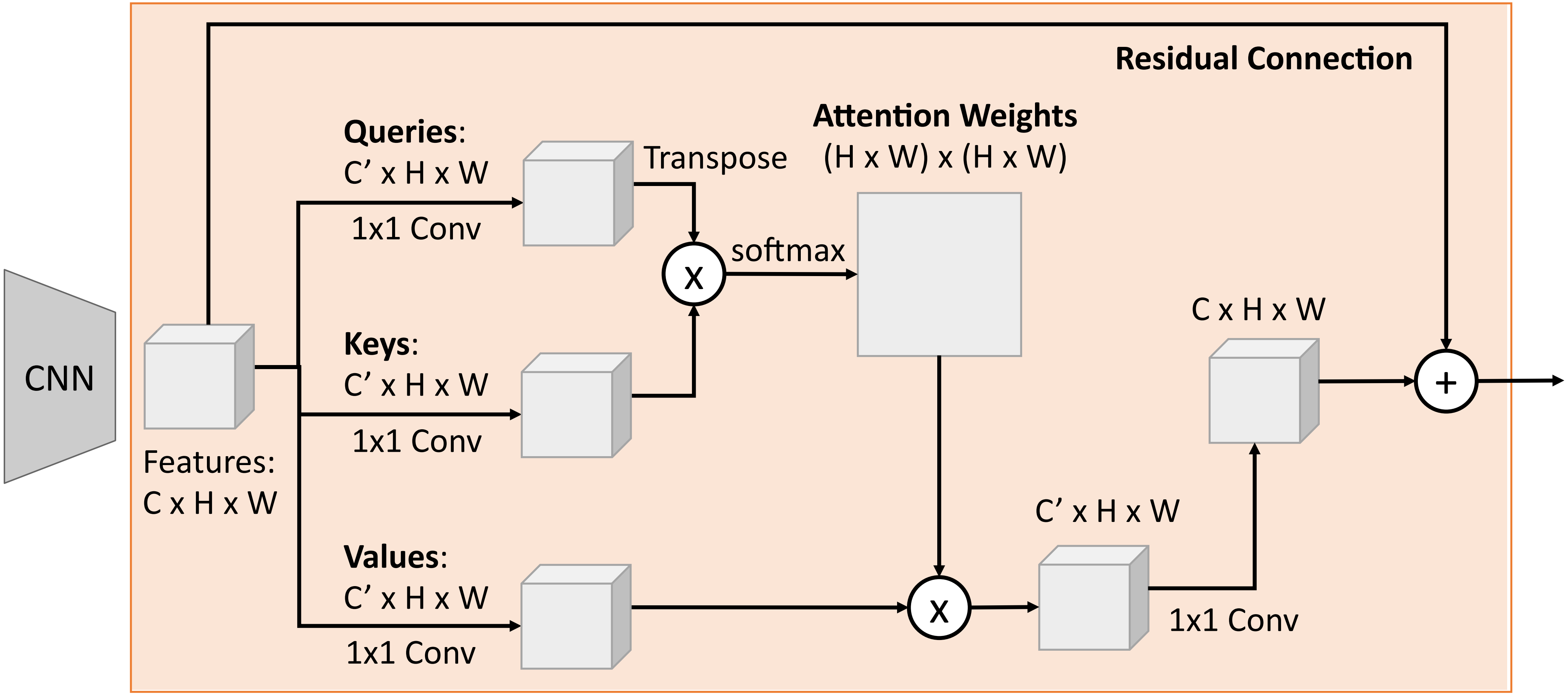


Zhang et al, “Self-Attention Generative Adversarial Networks”, ICML 2018

Example: CNN with Self-Attention

Input Image

Cat image is free to use under the Pixabay License

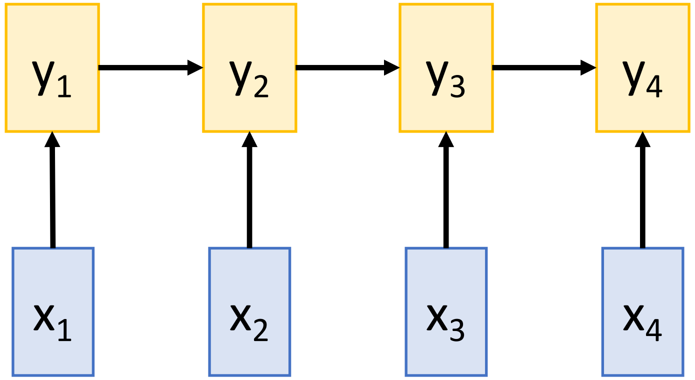


Self-Attention Module

Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2018

Three Ways of Processing Sequences

Recurrent Neural Network



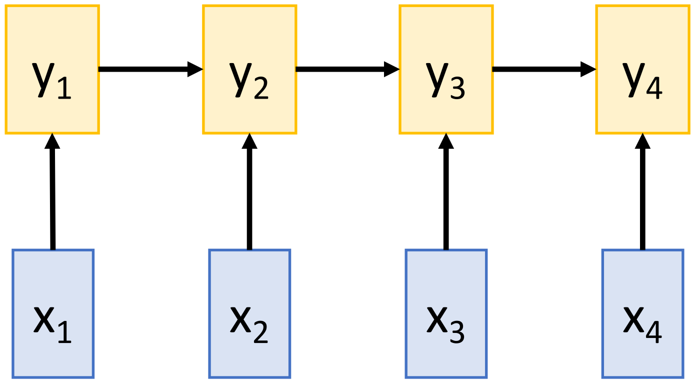
Works on **Ordered Sequences**

(+) Good at long sequences: After one RNN layer, h_T "sees" the whole sequence

(-) Not parallelizable: need to compute hidden states sequentially

Three Ways of Processing Sequences

Recurrent Neural Network

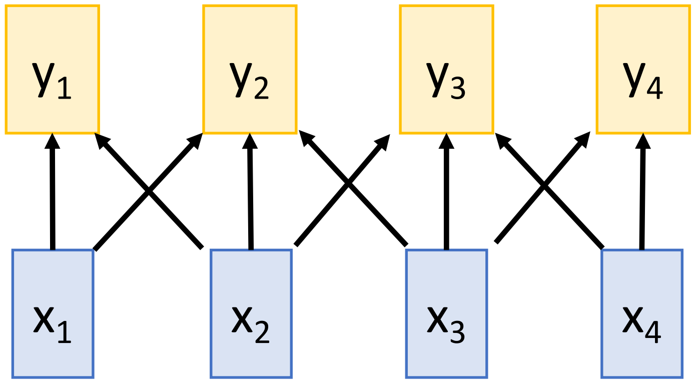


Works on **Ordered Sequences**

(+) Good at long sequences: After one RNN layer, h_T "sees" the whole sequence

(-) Not parallelizable: need to compute hidden states sequentially

1D Convolution



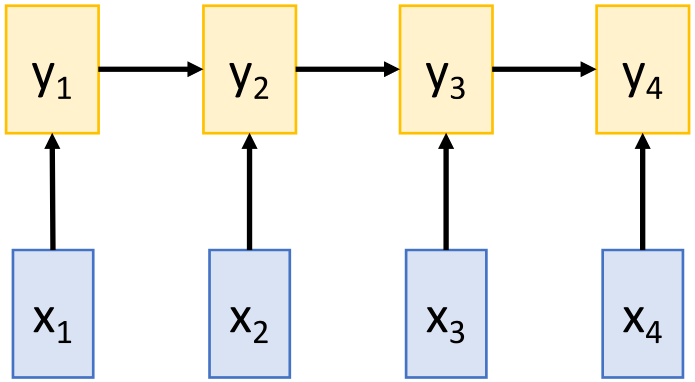
Works on **Multidimensional Grids**

(-) Bad at long sequences: Need to stack many conv layers for outputs to "see" the whole sequence

(+) Highly parallel: Each output can be computed in parallel

Three Ways of Processing Sequences

Recurrent Neural Network

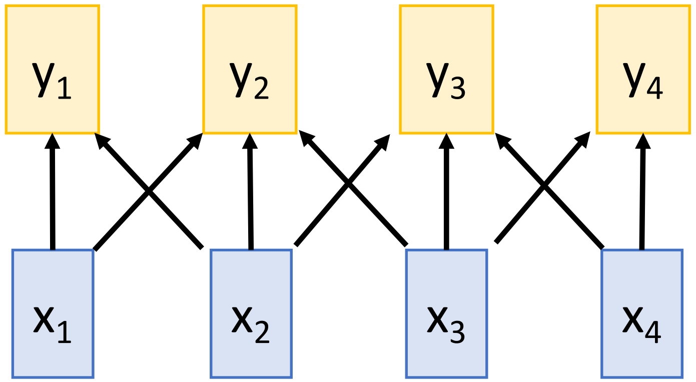


Works on **Ordered Sequences**

(+) Good at long sequences: After one RNN layer, h_T ”sees” the whole sequence

(-) Not parallelizable: need to compute hidden states sequentially

1D Convolution

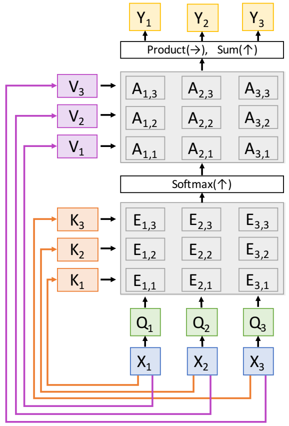


Works on **Multidimensional Grids**

(-) Bad at long sequences: Need to stack many conv layers for outputs to “see” the whole sequence

(+) Highly parallel: Each output can be computed in parallel

Self-Attention



Works on **Sets of Vectors**

(-) Good at long sequences: after one self-attention layer, each output “sees” all inputs!

(+) Highly parallel: Each output can be computed in parallel

(-) Very memory intensive

Three Ways of Processing Sequences

Recurrent Neural Network | 1D Convolution | Self-Attention

Attention is all you need

Vaswani et al, NeurIPS 2017

Recurrent Neural Network

Works on **Ordered Sequences**

(+) Good at long sequences: After one RNN layer, h_T "sees" the whole sequence

(-) Not parallelizable: need to compute hidden states sequentially

1D Convolution

Works on **Multidimensional Grids**

(-) Bad at long sequences: Need to stack many conv layers for outputs to "see" the whole sequence

(+) Highly parallel: Each output can be computed in parallel

Self-Attention

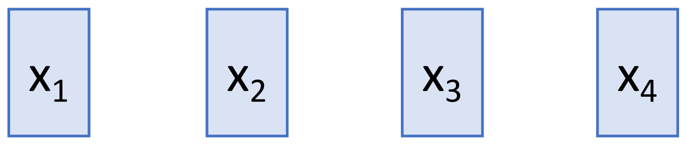
Works on **Sets of Vectors**

(-) Good at long sequences: after one self-attention layer, each output "sees" all inputs!

(+) Highly parallel: Each output can be computed in parallel

(-) Very memory intensive

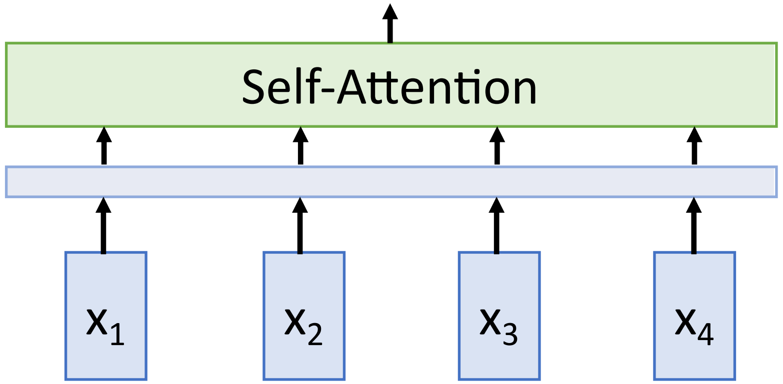
The Transformer



Vaswani et al, “Attention is all you need”, NeurIPS 2017

The Transformer

All vectors interact with each other

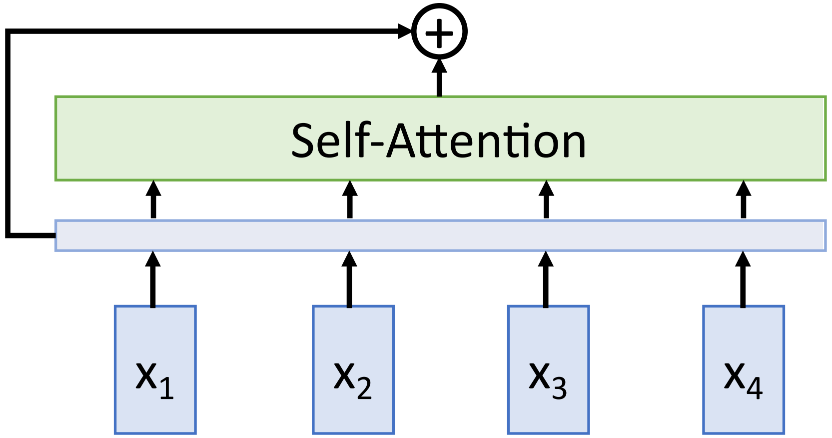


Vaswani et al, "Attention is all you need", NeurIPS 2017

The Transformer

Residual connection

All vectors interact with each other



Vaswani et al, "Attention is all you need", NeurIPS 2017

The Transformer

Recall **Layer Normalization**:

Given $h_1, \ldots, h_N$ (Shape: D)

scale: γ (Shape: D)

shift: β (Shape: D)

$\mu_i = (\sum_j h_{i,j})/D$ (scalar)

$\sigma_i = (\sum_j (h_{i,j} - \mu_i)^2/D)^{1/2}$ (scalar)

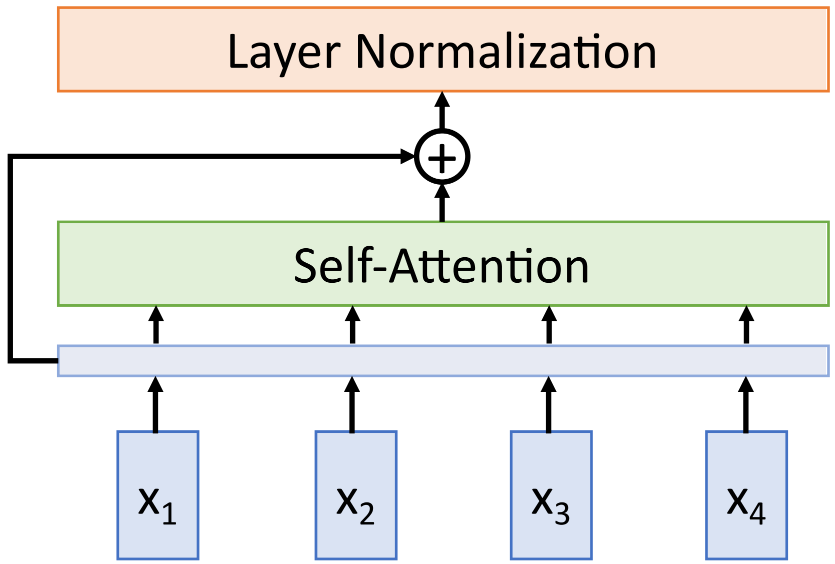
$z_i = (h_i - \mu_i) / \sigma_i$

$y_i = \gamma * z_i + \beta$

Ba et al, 2016

Residual connection

All vectors interact with each other



Vaswani et al, "Attention is all you need", NeurIPS 2017

The Transformer

Recall **Layer Normalization**:

Given $h_1, \ldots, h_N$ (Shape: D)

scale: γ (Shape: D)

shift: β (Shape: D)

$\mu_i = (\sum_j h_{i,j})/D$ (scalar)

$\sigma_i = (\sum_j (h_{i,j} - \mu_i)^2/D)^{1/2}$ (scalar)

$z_i = (h_i - \mu_i) / \sigma_i$

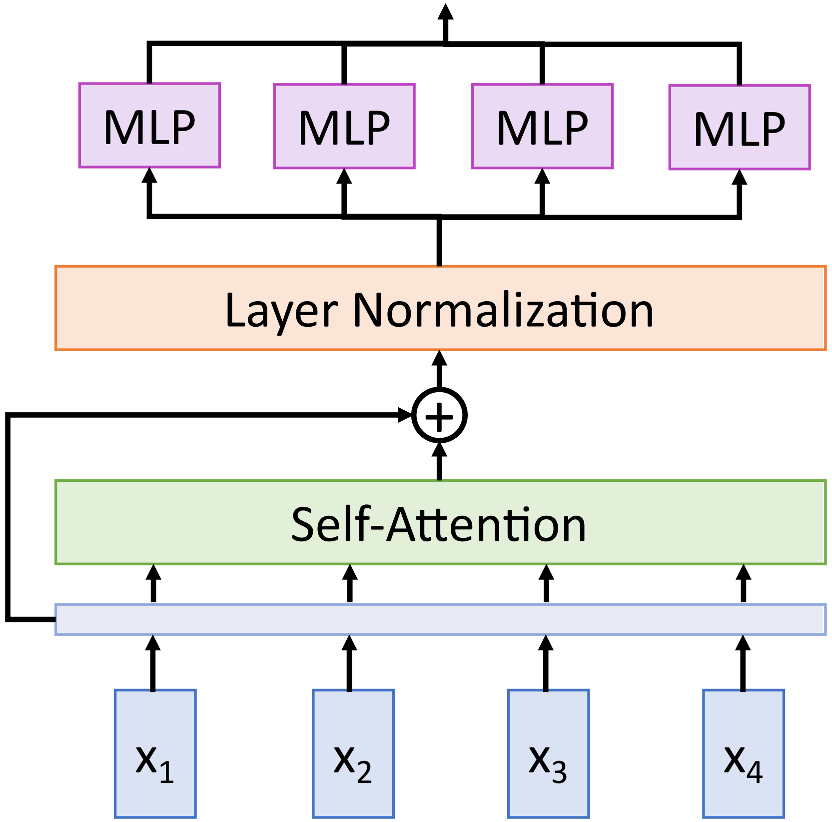
$y_i = \gamma * z_i + \beta$

Ba et al, 2016

MLP independently on each vector

Residual connection

All vectors interact with each other



Vaswani et al, "Attention is all you need", NeurIPS 2017

The Transformer

Recall **Layer Normalization**:

Given $h_1, \ldots, h_N$ (Shape: D)

scale: γ (Shape: D)

shift: β (Shape: D)

$\mu_i = (\sum_j h_{i,j})/D$ (scalar)

$\sigma_i = (\sum_j (h_{i,j} - \mu_i)^2/D)^{1/2}$ (scalar)

$z_i = (h_i - \mu_i) / \sigma_i$

$y_i = \gamma * z_i + \beta$

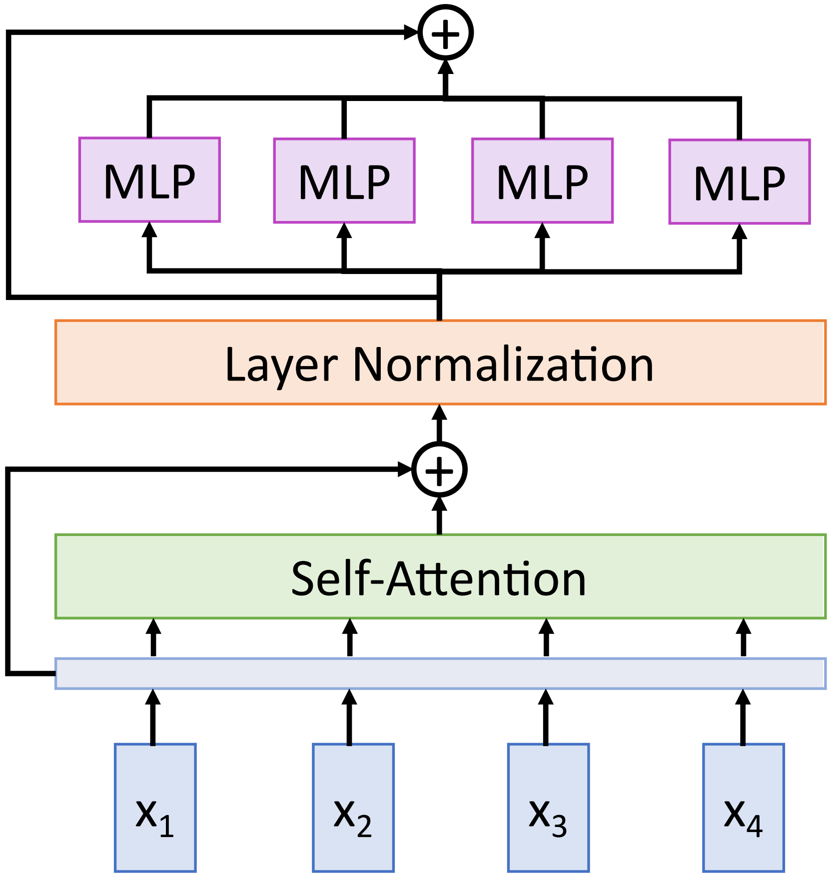
Ba et al, 2016

Residual connection

MLP independently on each vector

Residual connection

All vectors interact with each other



Vaswani et al, "Attention is all you need", NeurIPS 2017

The Transformer

Recall **Layer Normalization**:

Given $h_1, \ldots, h_N$ (Shape: D)

scale: γ (Shape: D)

shift: β (Shape: D)

$\mu_i = (\sum_j h_{i,j})/D$ (scalar)

$\sigma_i = (\sum_j (h_{i,j} - \mu_i)^2/D)^{1/2}$ (scalar)

$z_i = (h_i - \mu_i) / \sigma_i$

$y_i = \gamma * z_i + \beta$

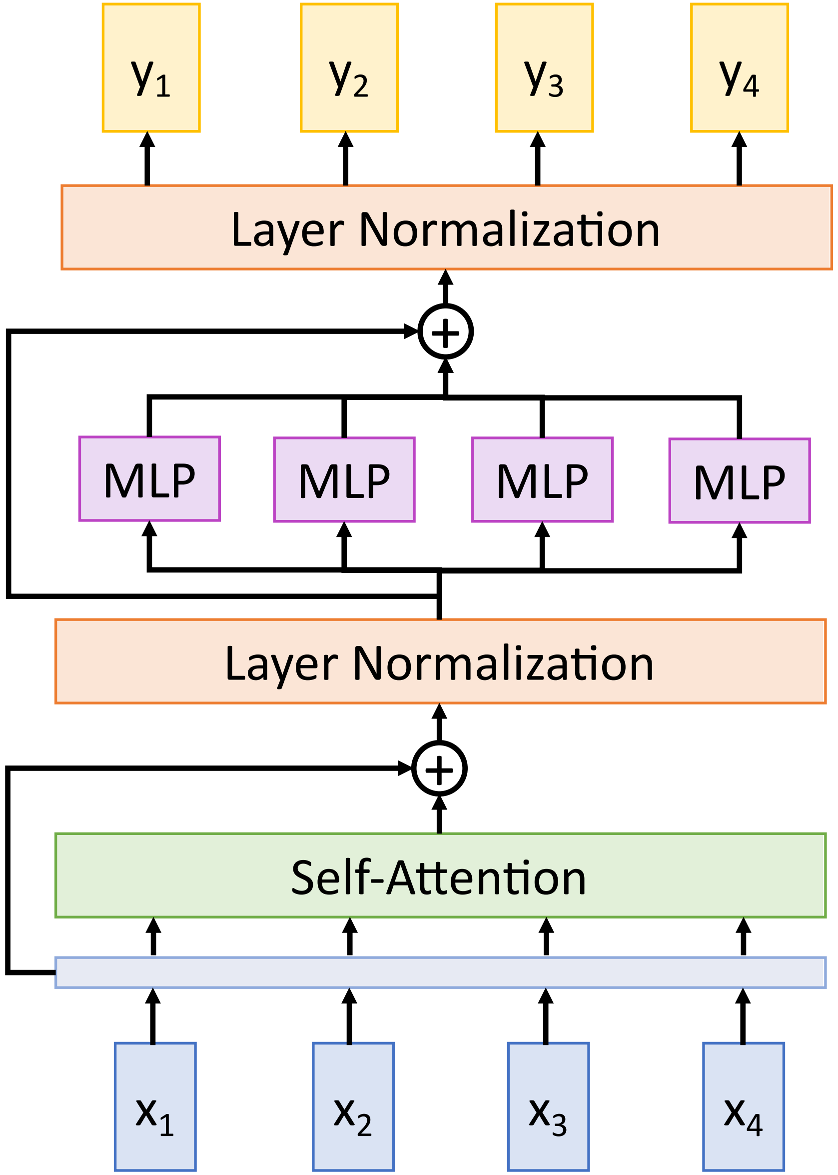
Ba et al, 2016

Residual connection

MLP independently on each vector

Residual connection

All vectors interact with each other



Vaswani et al, "Attention is all you need", NeurIPS 2017

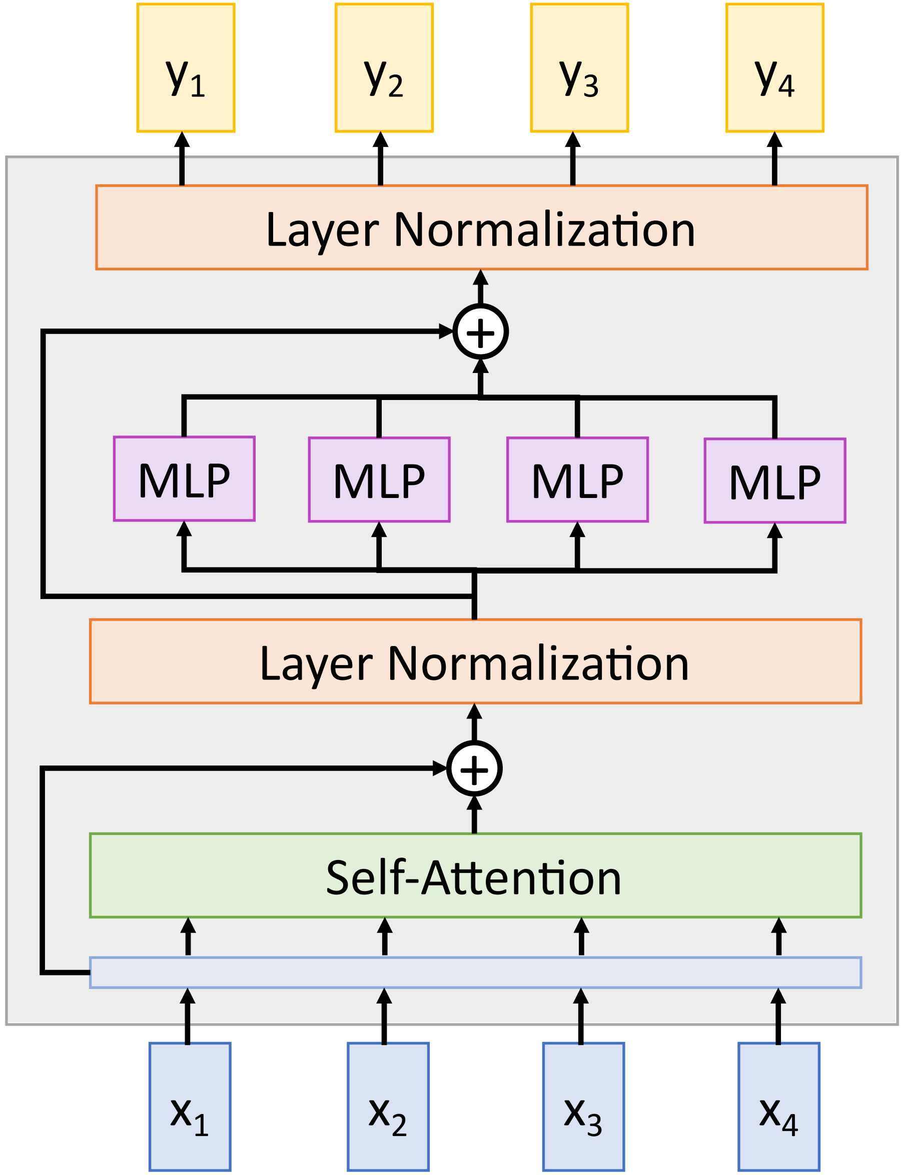
The Transformer

<u>**Transformer Block**</u>:
Input: Set of vectors x
Output: Set of vectors y

Self-attention is the only interaction between vectors!

Layer norm and MLP work independently per vector

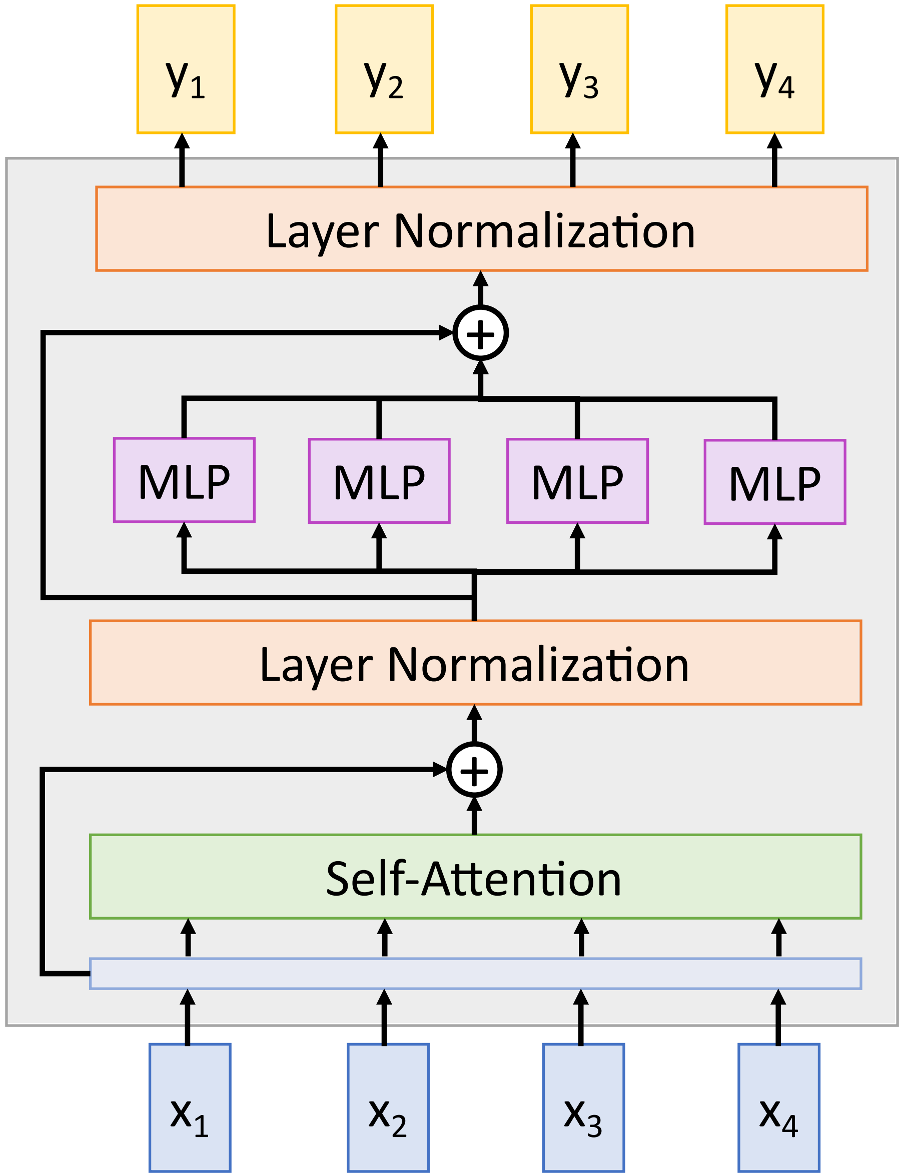
Highly scalable, highly parallelizable



Vaswani et al, “Attention is all you need”, NeurIPS 2017

Post-Norm Transformer

Layer normalization is **after** residual connections

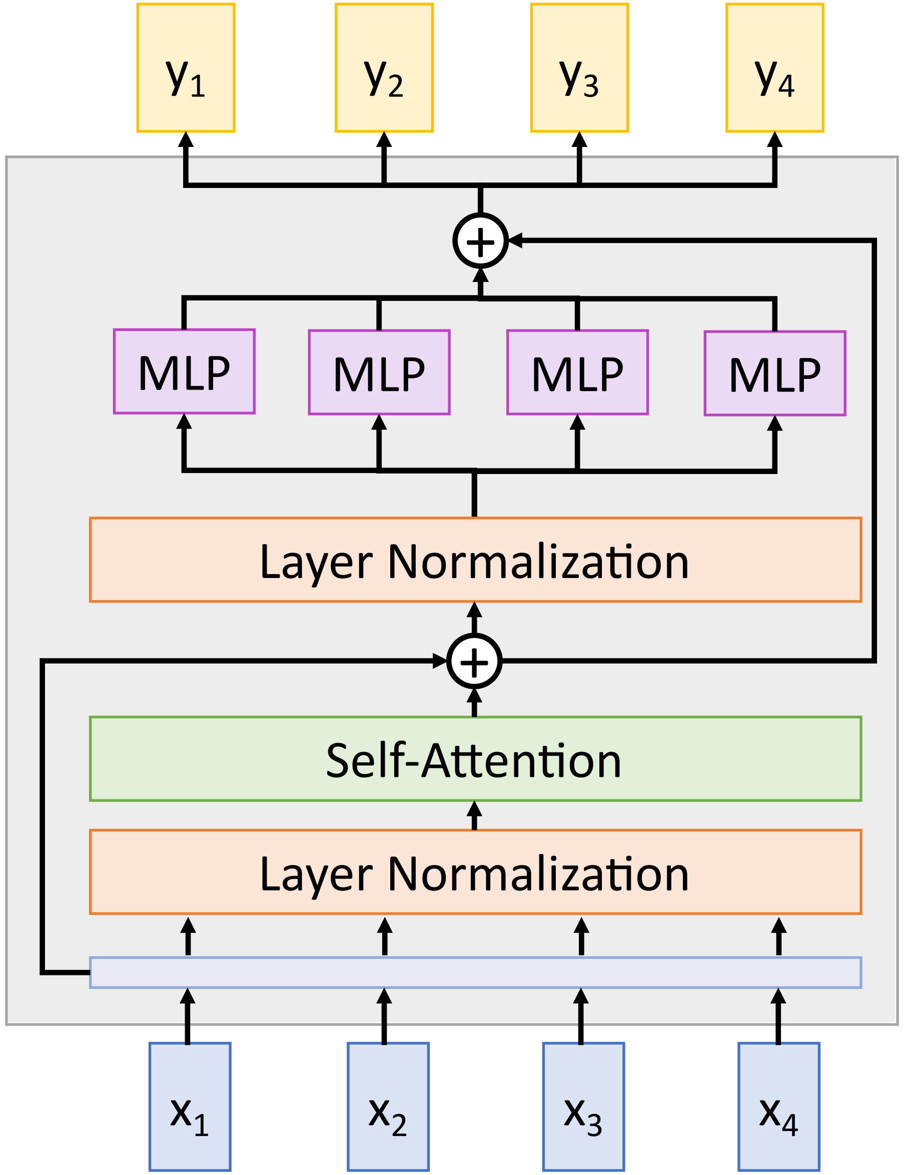


Vaswani et al, "Attention is all you need", NeurIPS 2017

Pre-Norm Transformer

Layer normalization is **inside** residual connections

Gives more stable training, commonly used in practice



Baevski & Auli, "Adaptive Input Representations for Neural Language Modeling", arXiv 2018

The Transformer

<u>Transformer Block</u>:
Input: Set of vectors x
Output: Set of vectors y

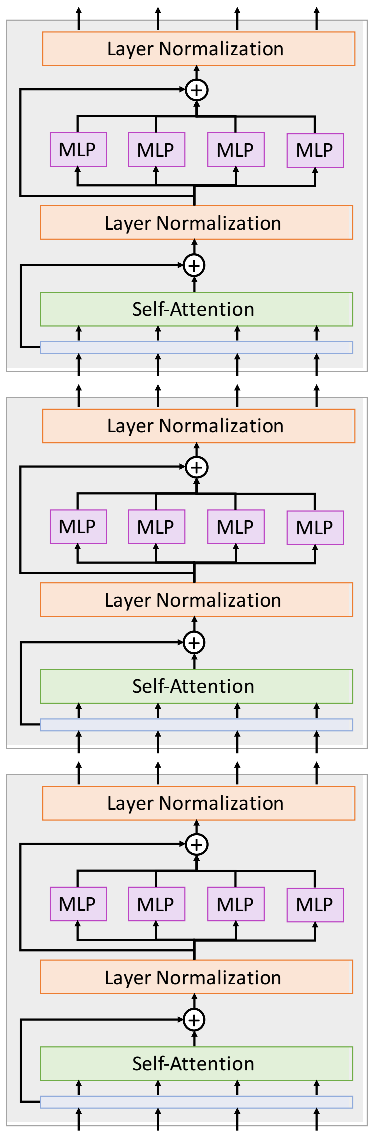
Self-attention is the only interaction between vectors!

Layer norm and MLP work independently per vector

Highly scalable, highly parallelizable

A **Transformer** is a sequence of transformer blocks

Vaswani et al:
12 blocks, D_Q=512, 6 heads



Vaswani et al, “Attention is all you need”, NeurIPS 2017

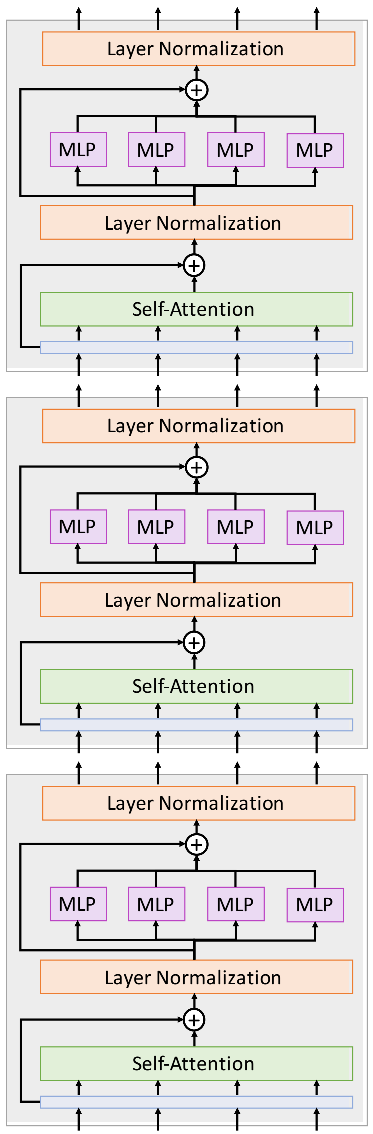
The Transformer: Transfer Learning

"ImageNet Moment for Natural Language Processing"

<u>Pretraining</u>:
Download a lot of text from the internet

Train a giant Transformer model for language modeling

<u>Finetuning:</u>
Fine-tune the Transformer on your own NLP task



Devlin et al, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", EMNLP 2018

Scaling up Transformers

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)

Vaswani et al, "Attention is all you need", NeurIPS 2017

Scaling up Transformers

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	

Devlin et al, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", EMNLP 2018

Scaling up Transformers

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	
XLNet-Large	24	1024	16	~340M	126 GB	512x TPU-v3 (2.5 days)
RoBERTa	24	1024	16	355M	160 GB	1024x V100 GPU (1 day)

Yang et al, XLNet: Generalized Autoregressive Pretraining for Language Understanding", 2019
Liu et al, "RoBERTa: A Robustly Optimized BERT Pretraining Approach", 2019

Scaling up Transformers

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	
XLNet-Large	24	1024	16	~340M	126 GB	512x TPU-v3 (2.5 days)
RoBERTa	24	1024	16	355M	160 GB	1024x V100 GPU (1 day)
GPT-2	48	1600	?	1.5B	40 GB	

Radford et al, "Language models are unsupervised multitask learners", 2019

Scaling up Transformers

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	
XLNet-Large	24	1024	16	~340M	126 GB	512x TPU-v3 (2.5 days)
RoBERTa	24	1024	16	355M	160 GB	1024x V100 GPU (1 day)
GPT-2	48	1600	?	1.5B	40 GB	
Megatron-LM	72	3072	32	8.3B	174 GB	512x V100 GPU (9 days)

Shoeybi et al, "Megatron-LM: Training Multi-Billion Parameter Languge Models using Model Parallelism", 2019

Scaling up Transformers

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	
XLNet-Large	24	1024	16	~340M	126 GB	512x TPU-v3 (2.5 days)
RoBERTa	24	1024	16	355M	160 GB	1024x V100 GPU (1 day)
GPT-2	48	1600	?	1.5B	40 GB	
Megatron-LM	72	3072	32	8.3B	174 GB	512x V100 GPU (9 days)
Turing-NLG	78	4256	28	17B	?	256x V100 GPU

Microsoft, "Turing-NLG: A 17-billion parameter language model by Microsoft", 2020

Scaling up Transformers

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	
XLNet-Large	24	1024	16	~340M	126 GB	512x TPU-v3 (2.5 days)
RoBERTa	24	1024	16	355M	160 GB	1024x V100 GPU (1 day)
GPT-2	48	1600	?	1.5B	40 GB	
Megatron-LM	72	3072	32	8.3B	174 GB	512x V100 GPU (9 days)
Turing-NLG	78	4256	28	17B	?	256x V100 GPU
GPT-3	96	12,288	96	175B	694GB	?

Brown et al, "Language Models are Few-Shot Learners", arXiv 2020

Scaling up Transformers

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	
XLNet-Large	24	1024	16	~340M	126 GB	512x TPU-v3 (2.5 days)
RoBERTa	24	1024	16	355M	160 GB	1024x V100 GPU (1 day)
GPT-2	48	1600	?	1.5B	40 GB	
Megatron-LM	72	3072	32	8.3B	174 GB	512x V100 GPU (9 days)
Turing-NLG	78	4256	28	17B	?	256x V100 GPU
GPT-3	96	12,288	96	175B	694GB	?
Gopher	80	16,384	128	280B	10.55 TB	4096x TPUv3 (38 days)

Rae et al, "Scaling Language Models: Methods, Analysis, & Insights from Training Gopher", arXiv 2021

Scaling up Transformers

$3,768,320 on Google Cloud (eval price)

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	
XLNet-Large	24	1024	16	~340M	126 GB	512x TPU-v3 (2.5 days)
RoBERTa	24	1024	16	355M	160 GB	1024x V100 GPU (1 day)
GPT-2	48	1600	?	1.5B	40 GB	
Megatron-LM	72	3072	32	8.3B	174 GB	512x V100 GPU (9 days)
Turing-NLG	78	4256	28	17B	?	256x V100 GPU
GPT-3	96	12,288	96	175B	694GB	?
Gopher	80	16,384	128	280B	10.55 TB	4096x TPUv3 (38 days)

Rae et al, "Scaling Language Models: Methods, Analysis, & Insights from Training Gopher", arXiv 2021

Generated Text from GPT-3

Bold text: Input prompt written by humans
Italics: Completion by GPT-3

Title: United Methodists Agree to Historic Split
Subtitle: Those who oppose gay marriage will form their own denomination
Article:

Brown et al, "Language Models are Few-Shot Learners", arXiv 2020

Generated Text from GPT-3

Bold text: Input prompt written by humans
Italics: Completion by GPT-3

Title: United Methodists Agree to Historic Split
Subtitle: Those who oppose gay marriage will form their own denomination
Article: *After two days of intense debate, the United Methodist Church has agreed to a historic split - one that is expected to end in the creation of a new denomination, one that will be "theologically and socially conservative," according to The Washington Post. The majority of delegates attending the church's annual General Conference in May voted to strengthen a ban on the ordination of LGBTQ clergy and to write new rules that will "discipline" clergy who officiate at same-sex weddings. But those who opposed these measures have a new plan: They say they will form a separate denomination by 2020, calling their church the Christian Methodist denomination. The Post notes that the denomination, which claims 12.5 million members, was in the early 20th century the "largest Protestant denomination in the U.S.," but that it has been shrinking in recent decades. The new split will be the second in the church's history. The first occurred in 1968, when roughly 10 percent of the denomination left to form the Evangelical United Brethren Church. The Post notes that the proposed split "comes at a critical time for the church, which has been losing members for years," which has been "pushed toward the brink of a schism over the role of LGBTQ people in the church." Gay marriage is not the only issue that has divided the church. In 2016, the denomination was split over ordination of transgender clergy, with the North Pacific regional conference voting to ban them from serving as clergy, and the South Pacific regional conference voting to allow them.*

GPT-3: Programming by prompt

Bold text: Input prompt written by humans
Italics: Completion by GPT-3

Poor English input: I eated the purple berries.
Good English output: I ate the purple berries.
Poor English input: Thank you for picking me as your designer. I'd appreciate it.
Good English output: Thank you for choosing me as your designer. I appreciate it.
Poor English input: The mentioned changes have done. or I did the alteration that you requested. or I changed things you wanted and did the modifications.
Good English output: The requested changes have been made. or I made the alteration that you requested. or I changed things you wanted and made the modifications.
Poor English input: I'd be more than happy to work with you in another project.
Good English output:

Brown et al, "Language Models are Few-Shot Learners", arXiv 2020

GPT-3: Programming by prompt

Input / Output examples
Test example
Italics: Completion by GPT-3

Poor English input: I eated the purple berries.
Good English output: I ate the purple berries.
Poor English input: Thank you for picking me as your designer. I'd appreciate it.
Good English output: Thank you for choosing me as your designer. I appreciate it.
Poor English input: The mentioned changes have done. or I did the alteration that you requested. or I changed things you wanted and did the modifications.
Good English output: The requested changes have been made. or I made the alteration that you requested. or I changed things you wanted and made the modifications.
Poor English input: I'd be more than happy to work with you in another project.
Good English output:

Brown et al, "Language Models are Few-Shot Learners", arXiv 2020

GPT-3: Programming by prompt

Input / Output examples
Test example
Italics: Completion by GPT-3

Poor English input: I eated the purple berries.
Good English output: I ate the purple berries.
Poor English input: Thank you for picking me as your designer. I’d appreciate it.
Good English output: Thank you for choosing me as your designer. I appreciate it.
Poor English input: The mentioned changes have done. or I did the alteration that you requested. or I changed things you wanted and did the modifications.
Good English output: The requested changes have been made. or I made the alteration that you requested. or I changed things you wanted and made the modifications.
Poor English input: I’d be more than happy to work with you in another project.
Good English output: *I'd be more than happy to work with you on another project.*

Brown et al, "Language Models are Few-Shot Learners", arXiv 2020

https://www.gwern.net/GPT-3#successes

GPT-3: Literary Style Transfer

Topic: humor: literary parodies and homages, pastiches, style parodies.
Parodies of the fantasy novel series *Harry Potter* in the style of various famous authors:
By Ernest Hemingway:
"It was a cold day on Privet Drive. A child cried. Harry felt nothing.

https://www.gwern.net/GPT-3#successes

GPT-3: Literary Style Transfer

Topic: humor: literary parodies and homages, pastiches, style parodies.
Parodies of the fantasy novel series *Harry Potter* in the style of various famous authors:
By Ernest Hemingway:
"It was a cold day on Privet Drive. A child cried. Harry felt nothing. He was dryer than dust. He had been silent too long. He had not felt love. He had scarcely felt hate. Yet the Dementor's Kiss killed nothing. Death didn't leave him less dead than he had been a second before. It wasn't about living or dying really. It wasn't about death. It was about the Dark Lord, his creator, his engenderer. He was not resurrected. He was created again. He was whole again in the courtyard of Malfoy Manor."

GPT-3: Literary Style Transfer

Topic: humor: literary parodies and homages, pastiches, style parodies.
Parodies of the fantasy novel series *Harry Potter* in the style of various famous authors:
By Ernest Hemingway:
"It was a cold day on Privet Drive. A child cried. Harry felt nothing. He was dryer than dust. He had been silent too long. He had not felt love. He had scarcely felt hate. Yet the Dementor's Kiss killed nothing. Death didn't leave him less dead than he had been a second before. It wasn't about living or dying really. It wasn't about death. It was about the Dark Lord, his creator, his engenderer. He was not resurrected. He was created again. He was whole again in the courtyard of Malfoy Manor."
By Jane Austen:
"It is a truth universally acknowledged, that a broken Harry is in want of a book. This he knows to his cost; pressing that wretched nose against the window of a bookstore in Diagon Alley the last thing before he goes into hiding for several years, where upon, he goes straight to Azkaban."

GPT-3: Literary Style Transfer

Topic: humor: literary parodies and homages, pastiches, style parodies.
Parodies of the fantasy novel series *Harry Potter* in the style of various famous authors:
By Ernest Hemingway:
"It was a cold day on Privet Drive. A child cried. Harry felt nothing. He was dryer than dust. He had been silent too long. He had not felt love. He had scarcely felt hate. Yet the Dementor's Kiss killed nothing. Death didn't leave him less dead than he had been a second before. It wasn't about living or dying really. It wasn't about death. It was about the Dark Lord, his creator, his engenderer. He was not resurrected. He was created again. He was whole again in the courtyard of Malfoy Manor."
By Jane Austen:
"It is a truth universally acknowledged, that a broken Harry is in want of a book. This he knows to his cost; pressing that wretched nose against the window of a bookstore in Diagon Alley the last thing before he goes into hiding for several years, where upon, he goes straight to Azkaban."
By Arthur Conan Doyle:
"Harry pushed at the swinging doors of the bookshop hard, and nearly knocked himself unconscious. He staggered in with his ungainly package, his cheeks scarlet with cold and the shame of having chosen the wrong month to go Christmas shopping. The proprietor of the store, however, didn't cast even a cursory glance at him, being far more interested in an enormous hunk of wood lying in the middle of the floor, which certainly looked like a gallows. Yes, the proprietor said to a reedy old man wearing a bowler hat and a forlorn expression that made Harry want to kick him, I can rent you such a gallows for a small fee."

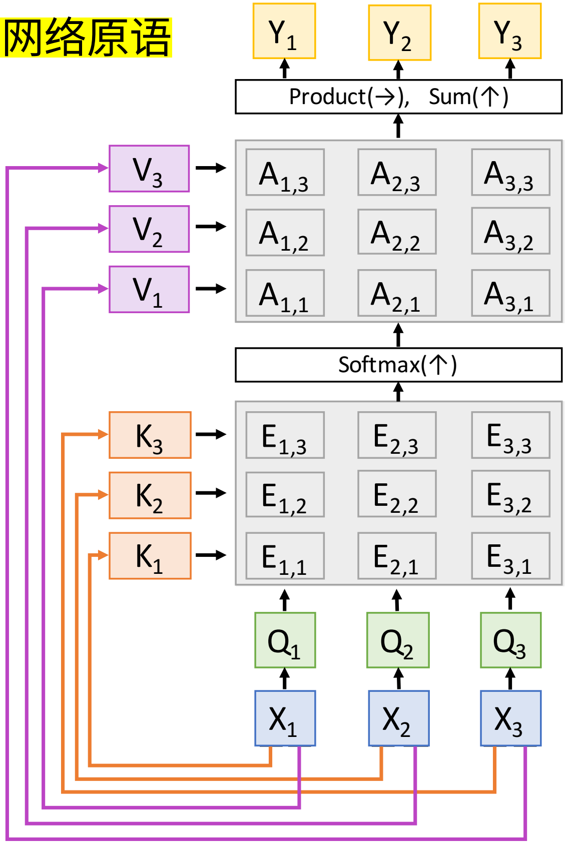
Summary

Adding **Attention** to RNN models lets them look at different parts of the input at each timestep

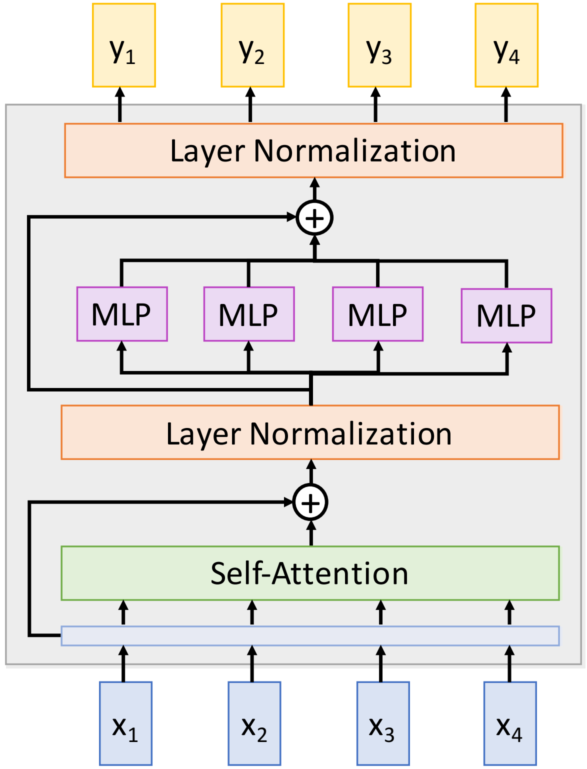
A dog is standing on a hardwood floor.

Generalized **Self-Attention** is new, powerful neural network primitive

神经网络原语



Transformers are a new neural network model that only uses attention



Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Next Time: Vision Transformers!